

Self-Assembly with Geometric Tiles

Bin Fu* Matthew J. Patitz† Robert T. Schweller‡ Robert Sheline§

Abstract

In this work we propose a generalization of Winfree’s abstract Tile Assembly Model (aTAM) in which tile types are assigned rigid shapes, or geometries, along each tile face. We examine the number of distinct tile types needed to assemble shapes within this model, the temperature required for efficient assembly, and the problem of designing compact geometric faces to meet given compatibility specifications. Our results show a dramatic decrease in the number of tile types needed to assemble $n \times n$ squares to $\Theta(\sqrt{\log n})$ at temperature 1 for the most simple model which meets a lower bound from Kolmogorov complexity, and $O(\log \log n)$ in a model in which tile aggregates must move together through obstacle free paths within the plane. This stands in contrast to the $\Theta(\log n / \log \log n)$ tile types at temperature 2 needed in the basic aTAM. We also provide a general method for simulating a large and computationally universal class of temperature 2 aTAM systems with geometric tiles at temperature 1. Finally, we consider the problem of computing a set of compact geometric faces for a tile system to implement a given set of compatibility specifications. We show a number of bounds on the complexity of geometry size needed for various classes of compatibility specifications, many of which we directly apply to our tile assembly results to achieve non-trivial reductions in geometry size.

*Department of Computer Science, University of Texas - Pan American, binfu@cs.panam.edu

†Department of Computer Science, University of Texas - Pan American, mpatitz@cs.panam.edu

‡Department of Computer Science, University of Texas - Pan American, schweller@cs.panam.edu

§Department of Computer Science, University of Texas - Pan American, b.sheline@gmail.com

1 Introduction

The stunning diversity of biological tissues and structures found in nature, including examples such as signaling axons stretching from neurons, powerfully contracting muscle tissue, and specifically tailored coats protecting viral payloads, are composed of basic molecular building blocks called proteins. These proteins, in turn, are assembled from an amazingly small set of only around 20 amino acids. So how is it that so much structural and functional variety can be derived from so few unique components? The simplified answer is “geometry”. Essentially, a protein’s function is determined by its 3-dimensional shape, or geometry. The exact sequence of amino acids which compose a protein (along with environmental influences such as temperature and pH levels) determine how that particular string of amino acids will fold into a protein’s characteristic 3-dimensional structure. However, as simple as it may sound, the resulting geometries are often extremely complex, and predicting them has proven to be computationally intractable. It is from such geometrically intricate structure that nearly all of the complexity of life as we know it arises.

Scientists and inventors have always recognized nature as providing invaluable examples and inspiration, and as for many other fields, this is also true for the study of artificial self-assembling systems. Self-assembling systems are systems in which sets of relatively simple components begin in disconnected and disorganized initial states, and then spontaneously and autonomously combine to form more complex structures. Self-assembling systems are pervasive in nature, and their power for creating intricate structures at even the nano-scale have inspired researchers to design artificial systems which self-assemble. One such productive line of research has followed from the introduction of the Tile Assembly Model (TAM) by Winfree in [27]. As a basic model, the TAM has proven powerful, providing a basis for laboratory implementations [5, 7, 16, 19, 20, 23, 24, 29] as well as copious amounts of theoretical work [6, 9, 11, 12, 18, 25, 26, 28]. However, in this work, we have once again looked to the guidance provided by nature, this time in terms of the power and importance of the geometric complexity of the components of self-assembling systems, to extend the TAM in an attempt to harness that power.

1.1 Overview

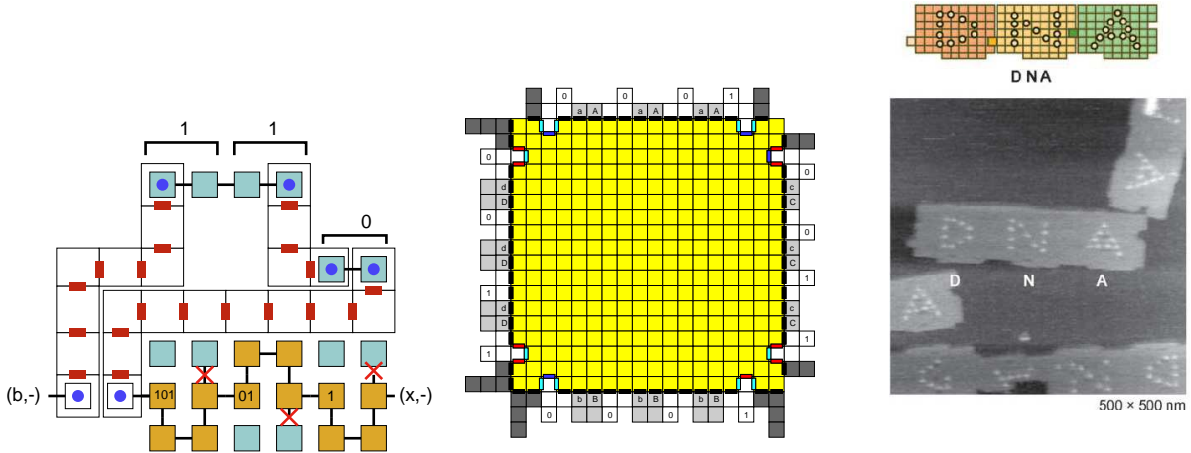


Figure 1: The use of jigsaw faced macro tiles for self-assembly is emerging in both theoretical and experimental work. This figure contains three separate recent examples. The first figure depicts the theoretical technique of encoding binary strings within the geometry of tile growth into the third dimension, as seen by small blue tiles in this figure [9]. The second figure depicts a macro tile assembled using staged assembly from smaller tile types [10]. Finally, the third figure depicts the experimental work of [14] in which a jigsaw geometry on the face of tiles is created with the DNA origami technique.

$n \times n$ square	Tile Types	Temperature	Geometry Size
ATAM (previous work) [3, 22]	$\Theta(\log n / \log \log n)$	2	-
GTAM (Thms. 3.1, 3.3)	$\Theta(\sqrt{\log n})$	1	$O(\sqrt{\log n})$
2GAM (Thm. 4.1)	$O(\log \log n)$	2	$O(\log n \log \log n)$

Zig-zag simulation	Tile Type Scale	Glues	Temperature	Geometry Size
Theorem 3.5	$O(1)$	$O(\sigma_w)$	1	$\log \sigma_n + \log \log \sigma_n + O(1)$
Theorem 3.6	$O(1)$	1	1	$\log \sigma + \log \log \sigma + O(1)$

Compact Geometry Design	Geometry Size	Run Time
Random Matrix (Thms. G.5)	$L(M) = \Theta(n)$	
Diagonal 1's (Cor. G.4)	$L(M) = n$	$O(n^2)$
Diagonal 0's (Cor. G.9)	$\log n + 1 \leq L(M) \leq \log n + \log \log n$	$O(n^2)$
Ind. Sub. Matrices (Thm. G.18)	$L(M) = \sum L(M_i)$	
Ind. Sub. Matrices (Thm. G.18)	$L(M) \leq (1 + \epsilon) \max(\min(m_i, n_i)) + O(\log(n + m))$	$O((m + n)^3)$
Ind. Sub. Matrices (Thm. G.18)	$\max(L(M_i)) \leq L(M) \leq (1 + \epsilon) \max(L(M_i)) + O(\log(m + n))$	$O(2^{\max(L(M_i)) \min(m, n)} (m + n)^3)$

Bar to Bump Reduction	Geometry Size
Theorem F.3, F.2	n

Table 1: Summary of our Results. σ denotes the number of distinct glues of a tile system to be simulated, with σ_n and σ_w denoting only the number of north/south and west/east glue types respectively. $L(M)$ is the size of the smallest geometry that can satisfy a binary $n \times m$ compatibility matrix M .

We introduce a generalization of the abstract Tile Assembly Model (aTAM) in which tile types are assigned rigid shapes, or geometries, along each tile face. This model is motivated by the plausibility of implementing novel sophisticated nanoscale shapes with technology such as DNA origami [21]. We show that this model permits substantially greater efficiency in terms of tile type complexity when compared to assembling shapes in the basic temperature 2 aTAM. Furthermore, these efficiency improvements hold even at temperature 1.

1.2 Results

The abstract tile assembly model (aTAM) [27], as well as many of the nanoscale self-assembly models spawned by it, feature single stranded DNA sequences as the primary mechanism for decision making. This commonality applies to weak systems such as deterministic temperature-1 assembly, as well as stronger ones that rely on higher temperatures or stochastic methods. Since it is known that DNA strands are capable of hybridizing with sequences other than their exact Watson-Crick compliments, it is therefore reasonable to consider a tile assembly model in which one glue can potentially bond with an arbitrary subset of the other glues, with possibly differing strengths. Aggarwal et. al. [8] have shown that such a *non-diagonal* glue function allows for significant efficiency gains in terms of the numbers of unique tiles used to assemble a target shape. Despite this potentially promising result, it is also true that designing non-specific hybridization pairs, while possible, is severely limited in a practical sense, and would likely introduce a potential for error in a much greater sense than is already present in laboratory experiments.

If non-specific binding is impractical or impossible to implement, but powerful in theory, the question remains: are there any other mechanisms by which this power can be realized? One possible answer to this question is motivated by advances in DNA origami [14, 21] in which DNA strands can be folded into blocks with semi-rigid jig-saw faces (see the rightmost image in Figure 1). In this work we introduce a generalization of the aTAM in which tile faces are given some rigid shape (which we hereon refer to as geometry). As suggested in Figure 2, the *geometric hindrance* which can be provided by this geometry is capable of simulating non-diagonal glue functions by creating a set of compatible and non-compatible

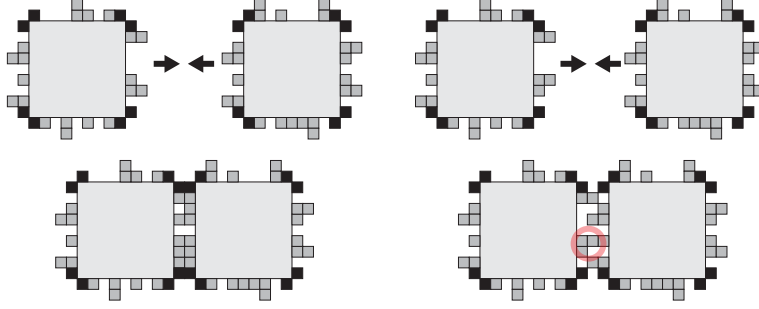


Figure 2: Examples of geometric tiles. Note that only the black portions on the corners are binding surfaces with glues, while the “teeth” in between provide potential geometric hindrance. Left: Compatible tiles. Right: Incompatible tiles (colliding teeth, which prevent the glue pads from coming together, are circled).

faces. We show that this new model realizes much of the power of non-specific hybridization. Among our results, we show that $n \times n$ squares can be assembled in $\Theta(\sqrt{\log n})$ distinct tile types, which meets an information theoretic lower bound for the model and improves what is possible without geometric tiles from $\Theta(\log n / \log \log n)$ (see [22]). In addition, this tile efficient construction requires only a temperature threshold of 1, thus showing this model can mimic both non-specific glue functions and temperature 2 self-assembly simultaneously.

Next, we show that temperature-1 systems utilizing geometry can efficiently simulate a powerful class of temperature-2 aTAM systems. This class of systems, called *zig-zag* systems, is capable of simulating arbitrary Turing machines and therefore universal computation. Furthermore, the simulation performed using geometric tiles is efficient in that it requires no increase in tile complexity (i.e. the number of unique tile types required) or in the size of the assembly. This is especially notable due to the fact that it is conjectured (although currently unproven) that temperature-1 systems in the aTAM are not computationally universal (see [13, 15] for more discussion about temperature-1 assembly in the aTAM).

While tile geometries provide a method for greatly reducing the tile complexity required to build squares in a seeded model similar to the aTAM (i.e. one in which tiles can only combine with a growing assembly one at a time), our next result holds for geometric tiles considered within the 2-handed assembly model (sometimes referred to by other names) [1, 2, 4, 8, 10, 18, 28]. We show that, in this model, the tile complexity required to build a square is reduced to only $O(\log \log n)$ tile types. The construction presented utilizes the ability of 2-handed assembly to grow assemblies by the combination of sub-assemblies composed of groups of previously combined tiles, and, coupled with complex geometric patterns on the tile edges, forces assembling components to undergo intricate patterns of relative motion in order to combine with each other. The tile geometries required are, however, complex ($O(\log n \log \log n)$) and in a 2-dimensional model require disconnected components. We then show a simple extension to 3 dimensions which allows for connected components while retaining all other features.

Finally, we conduct a detailed analysis of problems related to computing necessary patterns for tile geometries given specifications of the desired compatibility matrices (i.e. the listings of which tile sides should be compatible and incompatible with each other), with the goal being to minimize the size of the necessary geometries (as well as the running time of the computations). They deal with designing tile face geometries as a subset of $Z_1 \times Z_l$. Their solutions help show the feasibility and limitations of geometric tile face designs. We show a number of lower and upper bounds related to variants of the problem, some of which are incorporated into the previously mentioned constructions.

1.3 Organization of this paper

The remainder of this paper is organized as follows. In Section 2 we describe and define the new models introduced here. In Section 3, we present our constructions and proofs related to the self-assembly of $n \times n$

squares using $\Theta(\sqrt{\log n})$ tile types, as well as the simulation of zig-zag, temperature-2 aTAM systems by temperature-1 systems with geometric tiles. Section 4 describes our construction which utilizes geometric tiles as well as 2-handed assembly to self-assemble $n \times n$ squares using $O(\log \log n)$ tile types. Additionally, there is a technical appendix which contains the majority of the proofs and construction details for the results presented in the previous sections, as well as the results related to computing compatibility matrices.

2 Model

In this section we define the basic *geometric tile assembly model* (GTAM) and the *two-handed planar geometric tile assembly model* (2GAM). We begin with an informal description of the aTAM. We then define the Geometric Tile Assembly Model (GTAM). The GTAM generalizes the aTAM [27] by adding a geometry to each tile face that may prevent two tiles from attaching.

2.1 Basics

A tile type is a unit square with four sides, each having a glue consisting of a label (a finite string) and strength (0, 1, or 2). We assume a finite set T of tile types, but an infinite number of copies of each tile type, each copy referred to as a tile. A supertile (a.k.a., assembly) is a positioning of tiles on the integer lattice \mathbb{Z}^2 . Two adjacent tiles in a supertile interact if the glues on their abutting sides are equal. Each supertile induces a binding graph, a grid graph whose vertices are tiles, with an edge between two tiles if they interact. The supertile is τ -stable if every cut of its binding graph has strength at least τ , where the weight of an edge is the strength of the glue it represents. That is, the supertile is stable if at least energy τ is required to separate the supertile into two parts. A seeded tile assembly system (TAS) is a triple $T = (T, \tau, s)$, where T is a finite tile set, τ is the temperature, usually 1 or 2, and $s \in T$ is a special tile type denoted as the *seed*. Given a TAS $T = (T, \tau, s)$, a supertile is producible if either it is the seed tile, or it is the τ -stable result of attaching a single tile $r \in T$ to a producible supertile. A supertile α is terminal if for every tile type $r \in T$, r cannot be τ -stably attached to α . A TAS is directed (a.k.a., deterministic or confluent) if it has only one terminal, producible supertile. Given a connected shape $X \subset \mathbb{Z}^2$, a TAS T produces X uniquely if every producible, terminal supertile places tiles only on positions in X (appropriately translated if necessary).

2.2 Geometric Tiles and the Basic Geometric Tile Assembly Model (GTAM)

In this paper we generalize the basic aTAM by assigning a geometric pattern to each side of a tile type along with its glue. For each tile set in the GTAM, fix two values $w, \ell \in \mathbb{N}$. While at a high-level we still consider tiles as occupying unit squares within the plane, in order to determine whether or not adjacent tiles are *geometrically compatible* with each other, we define a *tile body* to be an $\ell \times \ell$ square (see Figure 3), and we define a (*tile face*) *geometry* to be a subset of $\mathbb{Z}_w \times \mathbb{Z}_\ell$. A *geometric* tile type consists of a tile body which has both a glue and a geometry assigned to each side. For a tile type t , let $northGeometry(t)$ denote the geometry assigned to the north side of t . Define $eastGeometry(t)$, $southGeometry(t)$, and $westGeometry(t)$ analogously. Intuitively, the geometry of a tile type face represents the positions of inflexible bumps, or “filled-in” locations of the $w \times \ell$ rectangle, that can prevent two tiles from lining up adjacently to one another so that the rectangles of their adjacent geometries completely overlap. Only if the $w \times \ell$ geometries on adjacent sides of two combining tiles can completely overlap so that no location contains a filled-in portion of both, can any glues on those adjacent sides interact. Formally, we say a tile type t is *east incompatible* with tile type r if $eastGeometry(t) \cap westGeometry(r) \neq \emptyset$. We define *north*, *south*, and *west* incompatibility analogously. Seeded Geometric Tile Assembly takes place in the same manner as in the aTAM, with the added requirement that a tile type cannot be attached to a supertile at a position in which the tile type is either east, west, north, or south incompatible with another adjacent tile type in the supertile at a position west, east, south, or north, respectively, of the attachment position. As in the original aTAM, tiles are not allowed to rotate and must always maintain their pre-specified orientation, even while moving into position to attach to an assembly.

2.3 Two-Handed Geometric Tile Assembly Model

The Two-Handed Geometric Tile Assembly Model (2GAM) extends the GTAM by allowing large assembled supertiles to attach to one another. We further restrict the model to planar assembly in which two supertiles may only attach if there exists a collision free path for the supertiles to traverse to reach their point of connection. In two dimensional assembly this enforces that supertiles must be able to slide into position while staying in the 2D plane. With standard aTAM tiles, this requirement enforces that individual tiles of a supertile do not collide with individual tiles from another supertile while the supertiles shift into position. With geometric tiles, we must also enforce that the geometries of individual tiles do not overlap with other tile geometries.

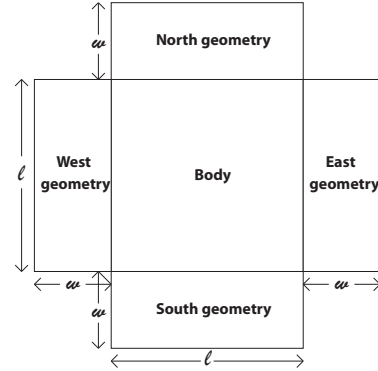


Figure 3: Definition of a geometric tile.

Informal Definition of the 2GAM As in the GTAM, tiles are composed of tile bodies and tile face geometries as shown in Figure 3.

Within the 2GAM, two tiles may attach if 1) there exists a collision free path within the 2D plane to shift the tiles into an adjacent position in which the east (or south) geometry box of one tile exactly overlaps the west (or north) geometry box of the second tile, and 2) the east (north) and west (south) glues of each tile are equal and have strength at least τ . More generally, preassembled multiple tile supertiles may come together if there is a collision free path in which the supertiles line up to create a τ -stable assembly. The set of *producible* supertiles within the 2GAM is defined recursively: As a base case, all singleton supertiles consisting of a single tile are producible. Recursively, for any two producible supertiles α and β such that there exists a collision free path within the plane to shift α and β into a τ -stable configuration γ , then the supertile γ is also producible. The subset of producible assemblies of a 2GAM system to which no producible assembly can attach defines the *terminally produced* supertiles. Intuitively, this set represents the set of assemblies we expect to see from a system if it is given enough time to assemble, and we refer to this as the output of the system. A 2GAM is directed (e.g., deterministic, confluent) if it has only one terminal, producible supertile. Given a connected shape $X \subseteq \mathbb{Z}^2$, a 2GAM Γ produces X uniquely if every producible, terminal supertile places tiles only on positions in X (appropriately translated if necessary).

Please refer to Section A for a more formal definition of the 2GAM model. Additionally, for a discussion of the different types of tile face geometries that are possible and the classes into which they can be categorized, please see Section B.

3 Complexities for the GTAM: Squares and $\tau = 1$ Assembly

In this section we examine the power of the GTAM in the context of efficiently building squares and simulating temperature $\tau = 2$ ATAM systems at $\tau = 1$. We first show in Section 3.1 that the tile complexity of $n \times n$ squares in the GTAM is $\Theta(\sqrt{\log n})$ for almost all n by providing an order $\sqrt{\log n}$ tile complexity upper bound construction for all n , and a matching information theoretic lower bound for almost all n . In addition, our upper bound construction utilizes only temperature $\tau = 1$. This stands in contrast to the temperature $\tau = 2$, $\Theta(\log n / \log \log n)$ tile complexity result that can be achieved in the ATAM [3].

As the square construction shows, the GTAM seems to be powerful at $\tau = 1$. In Section 3.2 we consider the problem of simulating $\tau = 2$ ATAM systems within the GTAM, but at $\tau = 1$. We show that for a large class of temperature $\tau = 2$ ATAM systems called *zig-zag* systems, such a simulation is possible with no scale up in tile complexity or assembly size. Of particular note is the fact that zig-zag systems are capable of simulating universal Turing machines, something that is conjectured to not be possible in the ATAM at $\tau = 1$.

3.1 The Tile Complexity of GTAM squares: $\Theta(\sqrt{\log n})$

In this section we analyze the size of the smallest tile type GTAM system that uniquely assembles an $n \times n$ square. Our first result is a construction that will assemble an $n \times n$ square using $O(\sqrt{\log n})$ tile types. We then show that this is tight for almost all n by applying an information theoretic argument to show that for almost all n , at least $\Omega(\sqrt{\log n})$ distinct tile types are required to uniquely assemble an $n \times n$ square. This result stands in contrast to the $\Theta(\log n / \log \log n)$ tile complexity for building squares in the standard ATAM model, showing that the GTAM is strictly more powerful than the ATAM. Further, our upper bound construction uses only temperature 1, while the ATAM construction requires temperature 2.

In the remainder of this section we prove the following theorems:

Theorem 3.1. The minimum tile complexity required to assemble an $n \times n$ square in the GTAM is $O(\sqrt{\log n})$. Further, this complexity can be achieved by a temperature $\tau = 1$ system with $O(\sqrt{\log n})$ size geometry.

Theorem 3.2. For almost all integers n , the minimum tile complexity required to assemble an $n \times n$ square in the GTAM is $\Omega(\sqrt{\log n})$.

For the sake of brevity, we only give a high level overview of the upper bound construction and place the details in referenced appendix sections.

3.1.1 $O(\sqrt{\log n})$ Construction Overview

The tile system for the assembly of $n \times n$ squares in the GTAM at temperature $\tau = 1$ and tile complexity $O(\sqrt{\log n})$ starts with the assembly of a roughly $2 \times \log n$ rectangle ($2 \times \lceil \log(\frac{n+1}{2}) \rceil + 2$ to be precise) that is used as a base to encode a roughly $\log n$ digit binary number ($\lceil \log(\frac{n+1}{2}) \rceil$ digits to be precise). The rectangle is efficiently built with $O(\sqrt{\log n})$ tile types by using a tile set for simulating a 2-digit, base- $\sqrt{\log n}$ counter. Such counters are known to exist in the ATAM at $\tau = 2$. To achieve $\tau = 1$ in the GTAM, we apply the transformation described in Theorem 3.6 to convert a zig-zag (see Definition 3.4) version of the $\tau = 2$ counter into a $\tau = 1$ GTAM system. The tileset for the basic $\tau = 2$ ATAM counter is given in Figure 10, and the $\tau = 1$ GTAM version is given in Figure 11. The details of this portion of the construction are described in Section C.0.2.

The next step of the construction grows a third row of tiles on top of the surface of the roughly $2 \times \log n$ rectangle. Within this assembled row one tile type representing a binary 0 or 1 bit is placed at each position, thus assembling a length roughly $\log n$ binary string upon completion. To generate $O(\log n)$ bits from only $O(\sqrt{\log n})$ distinct tile types is impossible in general within the ATAM. Within the GTAM, at this stage in the assembly we make use of the *non-specific* hindrance property of geometric tile faces to select the correct bit (and reject the wrong bit) at each of the $\log n$ bit positions. The key idea is that a collection of m geometric tile faces can be designed such that each face is compatible with a specified subset of the other faces, while incompatible with all others. Thus m tile faces can encode a compressed m^2 binary pieces of information (compatible or not compatible), thereby providing the possibility for a more tile type succinct assembly of an $n \times n$ square. A description of the decoder tile set is given in Figure 4 along with an example of how the assignment of geometry to tile faces permits the decoder tiles to efficiently select the correct bits. The details of this portion of the construction are described in Section C.0.3.

Once the binary string is assembled on the surface of the $3 \times \log n$ rectangle, we utilize a well known constant sized set of tiles that implement a binary counter in the ATAM at $\tau = 2$ [22]. This system reads a given surface of glues that denote an initial binary value for the counter, and then assembles upwards, incrementing a binary value encoded in tile types at every other row of the assembly. Once the counter is maxed out the construction stops, thus growing a rectangle of height roughly $2^{\log n}$ minus the initial value of the counter. This $\tau = 2$ ATAM counter construction is a zig-zag construction (see Definition 3.4). Thus, our construction applies Theorem 3.6 to convert to a $\tau = 1$ GTAM version.

Finally, with the ability to generate large length $O(n)$ rectangles with $O(\sqrt{\log n})$ tile types at $\tau = 1$, we combine 3 of these constructions to assemble the border of an $n \times n$ square using a factor of 3 times more tile types. A high level schematic of the approach is given in Figure 9. With the shell constructed, the

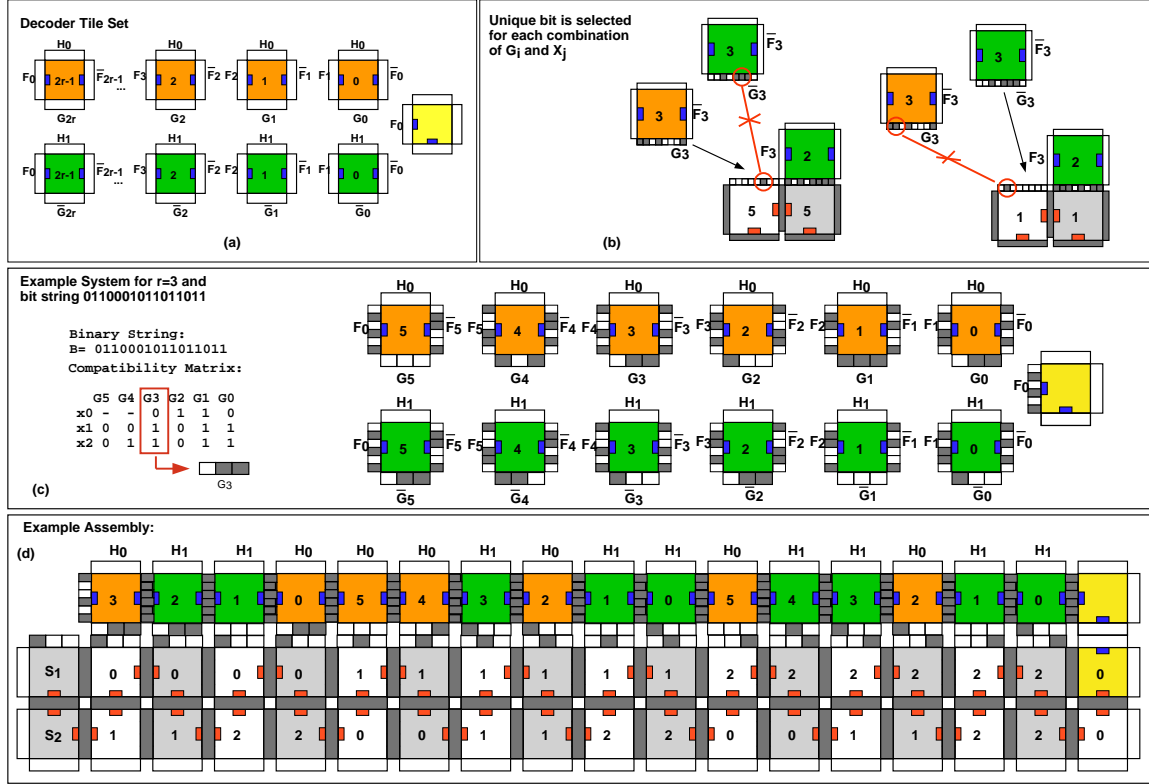


Figure 4: (a) The decoder set of tiles grows across the north face of the assembled counter from Figure 11 to build a binary string of tiles by placing at each position a tile type representing either a 0-bit or a 1-bit. (b) For a given binary string $B = b_{2r-1} \dots b_2 b_1 b_0$ to be assembled, the geometries G_j , \bar{G}_j , and X_i are assigned such that G_j and X_i are compatible if and only if $b_{2r(r-1)-2ri+j} = 0$. (c) For a given binary string $B = b_{2r-1} \dots b_2 b_1 b_0$ to be assembled, the listed compatibility matrix is obtained, along with the described assignment of geometry to each tile face. Such compatibility constraints can be achieved with geometry of length r for a length $2r^2 - 2$ string B . For more compact geometries in the case of less complex strings B , see Section G. (d) An example assembly of decoder tiles.

completion of the final rectangle can seed a growth of filler tiles to fill in the body of the square, finishing the construction. The final details of the construction are described in Section C.0.4.

3.1.2 Tight Kolmogorov Lower Bound for GTAM Squares: $\Omega(\sqrt{\log n})$

Theorem 3.3. For almost all integers n , the minimum tile complexity required to assemble an $n \times n$ square in the GTAM is $\Omega(\sqrt{\log n})$.

Proof. The Kolmogorov complexity of an integer n with respect to a universal Turing machine U is $K_U(n) = \min |p|$ s.t. $U(p) = b_n$ where b_n is the binary representation of n . It is known that $K_U(n) \geq \lceil \log n \rceil - \Delta$ for at least $1 - (\frac{1}{2})^\Delta$ of all n (see [17] for results on Kolmogorov complexity). Thus, for any $\epsilon > 0$, $K_U(n) \geq (1 - \epsilon) \log n = \Omega(\log n)$ for almost all n .

Consider a tile simulator program (of constant size in bits) that reads as input a GTAM tile system (encoded as a bit string). Suppose the simulator is modified so that it outputs the maximum extent (i.e. width or length) of a shape that is terminally produced by the input system. When such a simulator is paired with a GTAM system that uniquely assembles an $n \times n$ square, the combined program constitutes a program that outputs the integer n , implying that the total number of bits of the simulator (constant) plus the encoding of the tile set must be at least $K_U(n)$. As the simulator has a constant size, this implies that the number of bits to encode the GTAM system must be at least $K_U(n)$, which is $\Omega(\log n)$ for almost all n . To achieve our bound we now show that any GTAM system can be encoded using $O(|T|^2)$ bits (independent

of the size/area of the tile face geometries) assuming a constant bounded temperature. To achieve this, we do not explicitly encode the geometry for each tile face, but instead utilize a *compatibility matrix*.

Encoding a GTAM system. For a GTAM system $\Gamma = (T, \tau, s)$, arbitrarily index each distinct face of each distinct tile in T from 1 to $4|T|$. Define the *compatibility matrix* M for Γ to be the $4|T| \times 4|T|$ matrix such that $M_{i,j} = 1 \iff i$ is the index of an east (or north respectively) edge and j is the index of a west (south respectively) edge and i and j have incompatible edge geometries. M can be encoded using $O(T^2)$ bits, and the remaining portions of Γ can easily be encoded in asymptotically fewer bits, yielding an $O(T^2)$ bit encoding for any GTAM system. Note that even without the explicit representation of the GTAM's geometries, a simulator can derive what the system will build from the compatibility matrix M .

Now consider the smallest tile type GTAM system $\Gamma = (T, \tau, s)$ that uniquely assembles an $n \times n$ square. As Γ can be encoded in $O(|T|^2)$ bits, we know that for almost all n , $c_1|T|^2 \geq c_2 \log n$ for constants c_1, c_2 . Therefore, $|T| = \Omega(\sqrt{\log n})$ for almost all n . \square

3.2 Simulating Temperature $\tau = 2$ ATAM Systems with $\tau = 1$ GTAM Systems

Definition 3.4. Zig-Zag System. A tile system $\Gamma = (T, \tau, s)$ is called a zig-zag system if:

1. The location and type of the i^{th} tile to attach is the same for all assembly sequences.
2. The i^{th} tile attachment occurs to the north, west, or east (not south) of the previously placed tile attachment in all assembly sequences.

For the proofs of the following two theorems and technical details about the notion of tile system “simulation”, please see Section D.

Theorem 3.5. Any temperature $\tau = 2$ zig-zag ATAM tile system $\Gamma = (T, 2, s)$ can be simulated by a $\tau = 1$ GTAM tile system $\Upsilon = (R, 1, q)$ with tile type scale $|R|/|T| = O(1)$. The simulation utilizes geometry size at most $\log \sigma_n + \log \log \sigma_n + O(1)$ where σ_n is the number of distinct north/south glue types represented in T .

Theorem 3.6. Any temperature $\tau = 2$ zig-zag ATAM tile system $\Gamma = (T, 2, s)$ can be simulated by a $\tau = 1$ GTAM tile system $\Upsilon = (R, 1, q)$ using only 1 non-null glue type and tile type scale $|R|/|T| = O(1)$. The geometry size of the simulation system is at most $\log \sigma + \log \log \sigma + O(1)$ where σ is the number of distinct glue types represented in T .

4 2GAM Results

In this section, we explore the theoretical limits achievable when utilizing geometric tiles by designing tiles whose edges contain highly complex geometries. Furthermore, we move to the 2-handed variant of the GTAM, the 2GAM, to allow for the geometric hindrances experienced by individual tiles to be grouped and combined to provide more complex interactions between larger supertiles. The goal, rather than providing a realistic and potentially experimentally realizable set of constructions, is to gain further understanding into the interplay between geometry and the types of computations which can be carried out via algorithmic self-assembly.

We now present the details of our construction, which reduces the tile complexity required to self-assemble an $n \times n$ square to a mere $O(\log \log n)$ tile types, while requiring a geometry size of $O(\log n \log \log n)$. Our construction requires the constraint of planarity, in which components are not allowed to float into position from above or below the assembly, but must always be able to slide into position with a series of translations along only the x and y axes. However, the intricate geometric designs and complex series of movements require that individual tile geometries are composed of disconnected components. (Note that in Section E.5 we show how to extend the tiles into the third dimension, utilizing a total of 4 planes, in a manner which results in connected tiles and also implicitly enforces the restriction that only tile translations along the x and y axes must be sufficient to allow for tile attachments.)

4.1 Self-assembly of an $n \times n$ square with $O(\log \log n)$ tile types

Theorem 4.1. For every $n \in \mathbb{N}$, there exists a 2GAM tile system $\Gamma = (T, 2)$ which uniquely produces an $n \times n$ square, where $|T| = O(\log \log n)$, and with $O(\log n \log \log n)$ size geometry.

To prove Theorem 4.1, we present the following construction.

4.1.1 High-level sketch of the construction

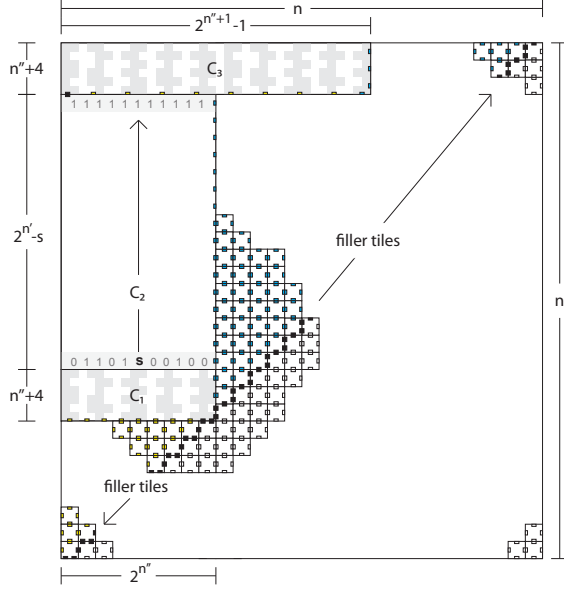


Figure 5: A high-level sketch of the construction for building a square in the 2GAM.

Following is a list of values based on the particular dimensions of the square to be formed and which are used throughout the following discussion:

- n : dimensions of the square to self-assemble
- n' : $\lceil \log n \rceil$
- n'' : $\lceil \log n' \rceil$
- s : $2^{n'} + 2^{n''} + 2n'' + 8 - n$
- h : $2^{n''-1} - 1$
- C_1 : 2-handed counter which counts from 0 through $2^{n''} - 1$ for a total of $2^{n''}$ columns
- C_2 : standard counter which counts from s through $2^{n'} - 1$ for a total of $2^{n'} - s$ columns
- C_3 : 2-handed counter with “buffer” columns which counts from 0 through $2^{n''+1} - 1$ for a total of $2^{n''+1} - 1$ columns

Figure 5 shows a high level view of the main components of this construction. Without loss

of generality, we can consider the construction to be composed of a series of sub-assemblies, or modules, which assemble in sequence, with each module completely assembling before the next begins. The careful design of all modules ensures that none can grow so that they occupy space required by another, and that each will be able to terminally grow to precisely defined dimensions that result in the final combination forming exactly an $n \times n$ square. For the rest of this discussion, we will describe the formation of the modules in such a sequence. (See Figure 12 for a series of high-level images which exemplify the ordering of the formation of the square from these modules.)

Similar to the construction in Section 3.1, this construction makes use of one counter, C_1 , to assemble an encoding of a number which in turn seeds another counter, C_2 . C_1 assembles in a 2-handed manner, meaning that each number which is counted is represented by exactly one one-tile-wide column of tiles, and individual columns form separately and then combine to form the full counter of length $2^{n''}$ (similar in design to counters found in [12]). Each column of the counter, besides representing a counter value, is used to represent (on the north face of the northernmost tile) one bit of the seed value s for C_2 . Each column can form in one of two versions: one that represents a 0, and one that represents a 1, for the corresponding bit of s . The east and west sides of the tiles forming the columns of this counter contain geometries which force the columns, in order to combine, to “wiggle” up and down in patterns based on the counter values of those columns. See Figure 6 for an example pair of compatible columns. The columns also contain tiles with geometries which “read” those patterns of wiggling and allow columns to combine with each other if

and only if they are the correct versions of the counter columns, namely those with the seed bit values which correctly correspond to their location in the counter. It is the tiles of this component as well as those of the counter C_3 to which the intricate geometries are applied, and thus they receive a much more detailed explanation in Section E.1.

C_2 is a standard binary counter (i.e. one that would also assemble correctly in the aTAM) which utilizes 16 tile types (see Figure 15) and grows to complete the majority of the western side of the square. Next, a small set of 7 “filler” tile types (see Figure 14b) fill in the majority of the square, and once they have filled in a sufficient portion of the northern portion they provide a platform to which C_3 can attach (as long as C_3 is fully formed). In order to provide a directed system with only one terminal assembly, the “incorrect” columns (those which couldn’t become part of C_1) are able to combine into the 2-handed counter structure C_3 via some extra buffer columns (see Figures 14 and 19). Finally, the filler tiles are able to complete the formation of the square. Note that the tile types which make up C_2 and the filler tiles require no geometries but only standard glues.

By utilizing the assembly of supertiles (i.e. sub-assemblies of grouped tiles) and carefully designing geometries which force the supertiles forming C_1 to move in well-defined patterns as they attach, we are able to essentially “transmit” information about tiles in one location of a supertile to the interfaces where potential binding is occurring with other tiles in the same supertile. By concatenating this information from such a group of distant tiles, the binding “decision” can be made based on an arbitrarily large amount of information (as long as the geometry sizes scale appropriately). This results in a dramatic lowering of the tile complexity required to assemble an $n \times n$ square, with the trade-off being an increase in the complexity of the tiles themselves.

Please see Section E for much more detail and several supplementary images describing this construction. Additionally, a comprehensive example has been provided in Section E.4 to which the reader can refer for additional clarity.

4.2 Analysis of tile complexity and geometry size

First, we analyze the tile complexity of this construction, module by module, in order to determine the overall complexity.

The tile complexity of each component is as follows:

- C_1 :

Counter tiles: There are n'' bit positions which each require a constant number of tile types (as can be seen from the depiction in Figure 14a), plus the requirement for a hard-coded column on each of the west and east sides, for a total of $O(\log \log n)$ tile types.

Cap tiles: There are 4 cap tile positions which each need to be able to represent a 0 or a 1 cap, for a total of 8 tile types.

- C_2 : 16 tile types.

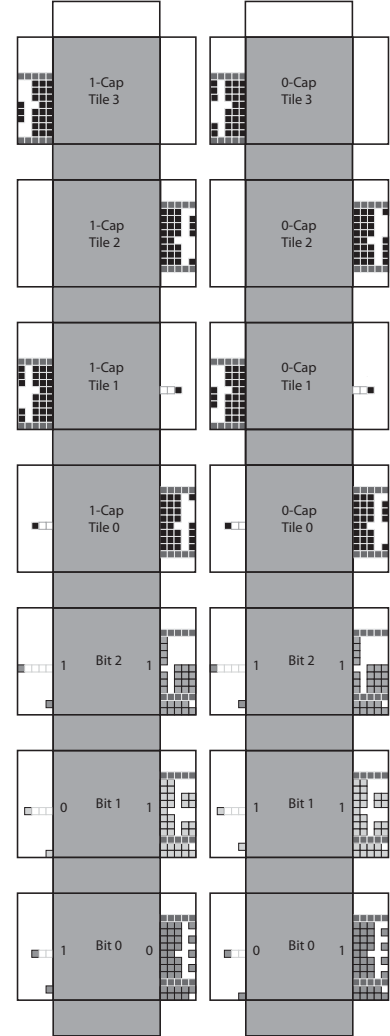


Figure 6: Example columns for the counter C_1 . Note that all colored areas are filled-in, and areas colored white are empty, although they may be outlined for reference.

- C_3 :

Counter tiles: There are n'' bit positions in the buffer columns which each require a constant number of tile types, for a total of $O(\log \log n)$ tile types.

Buffer cap tiles: There are 4 cap tile positions which each require a single tile type, for a total of 8 tile types.

- Filler tiles: 7 tile types.

Thus, the total tile type complexity is $O(\log \log n) + O(1) + O(1) + O(\log \log n) + O(1) + O(1) = O(\log \log n)$.

Next, we simply note that the geometries defined for all tiles in this construction consist of rectangles of dimensions $(2^{n''} + h + 4) \times (n'' + 2) = (2^{n''} + \lceil 2^{n''}/2 \rceil + 4) \times (n'' + 2) = O(\log n \times \log \log n)$, and therefore the geometry size is $O(\log n \log \log n)$.

References

- [1] Zachary Abel, Nadia Benbernou, Mirela Damian, Erik Demaine, Martin Demaine, Robin Flatland, Scott Kominers, and Robert Schweller, *Shape replication through self-assembly and RNase enzymes*, SODA 2010: Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms (Austin, Texas), Society for Industrial and Applied Mathematics, 2010.
- [2] Leonard Adleman, *Toward a mathematical theory of self-assembly (extended abstract)*, Tech. Report 00-722, University of Southern California, 2000.
- [3] Leonard Adleman, Qi Cheng, Ashish Goel, and Ming-Deh Huang, *Running time and program size for self-assembled squares*, Proceedings of the thirty-third annual ACM Symposium on Theory of Computing (New York, NY, USA), ACM, 2001, pp. 740–748.
- [4] Leonard Adleman, Qi Cheng, Ashish Goel, Ming-Deh Huang, and Hal Wasserman, *Linear self-assemblies: Equilibria, entropy and convergence rates*, In Sixth International Conference on Difference Equations and Applications, Taylor and Francis, 2001.
- [5] Robert D. Barish, Rebecca Schulman, Paul W. Rothmund, and Erik Winfree, *An information-bearing seed for nucleating algorithmic self-assembly*, Proceedings of the National Academy of Sciences **106** (2009), no. 15, 6054–6059.
- [6] Harish Chandran, Nikhil Gopalkrishnan, and John H. Reif, *The tile complexity of linear assemblies*, 36th International Colloquium on Automata, Languages and Programming, vol. 5555, 2009.
- [7] Ho-Lin Chen, Rebecca Schulman, Ashish Goel, and Erik Winfree, *Reducing facet nucleation during algorithmic self-assembly*, Nano Letters **7** (2007), no. 9, 2913–2919.
- [8] Qi Cheng, Gagan Aggarwal, Michael H. Goldwasser, Ming-Yang Kao, Robert T. Schweller, and Pablo Moisset de Espanés, *Complexities for generalized models of self-assembly*, SIAM Journal on Computing **34** (2005), 1493–1515.
- [9] Matthew Cook, Yunhui Fu, and Robert Schweller, *Temperature 1 self-assembly: Deterministic assembly in 3d and probabilistic assembly in 2d*, Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms, 2011.
- [10] Erik D. Demaine, Martin L. Demaine, Sándor P. Fekete, Mashhood Ishaque, Eynat Rafalin, Robert T. Schweller, and Diane L. Souvaine, *Staged self-assembly: nanomanufacture of arbitrary shapes with $O(1)$ glues*, Natural Computing **7** (2008), no. 3, 347–370.

- [11] David Doty, Jack H. Lutz, Matthew J. Patitz, Scott M. Summers, and Damien Woods, *Intrinsic universality in self-assembly*, Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science, 2009, pp. 275–286.
- [12] David Doty, Matthew J. Patitz, Dustin Reishus, Robert T. Schweller, and Scott M. Summers, *Strong fault-tolerance for self-assembly with fuzzy temperature*, Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2010), 2010, pp. 417–426.
- [13] David Doty, Matthew J. Patitz, and Scott M. Summers, *Limitations of self-assembly at temperature 1*, Theoretical Computer Science **412** (2011), 145–158.
- [14] Masayuki Endo, Tsutomu Sugita, Yousuke Katsuda, Kumi Hidaka, and Hiroshi Sugiyama, *Programmed-assembly system using DNA jigsaw pieces*, Chemistry: A European Journal (2010), 5362–5368.
- [15] Yunhui Fu and Robert Schweller, *Temperature 1 self-assembly: Deterministic assembly in 3d and probabilistic assembly in 2d*, Tech. Report 0912.0027, Computing Research Repository, 2009.
- [16] T.H. LaBean, E. Winfree, and J.H. Reif, *Experimental progress in computation by self-assembly of DNA tilings*, DNA Based Computers **5** (1999), 123–140.
- [17] M. Li and P. Vitanyi, *An introduction to komogorov complexity and its applications (second edition)*, Springer Verlag, New York, 1997.
- [18] Chris Luhrs, *Polyomino-safe DNA self-assembly via block replacement*, DNA14 (Ashish Goel, Friedrich C. Simmel, and Petr Sosík, eds.), Lecture Notes in Computer Science, vol. 5347, Springer, 2008, pp. 112–126.
- [19] Chengde Mao, Thomas H. LaBean, John H. Reif, and Nadrian C. Seeman, *Logical computation using algorithmic self-assembly of DNA triple-crossover molecules.*, Nature **407** (2000), no. 6803, 493–6.
- [20] John Reif, Sudheer Sahu, and Peng Yin, *Compact error-resilient computational DNA tiling assemblies*, DNA: International Workshop on DNA-Based Computers, LNCS, 2004.
- [21] Paul W. K. Rothmund, *Folding DNA to create nanoscale shapes and patterns*, Nature **440** (2006), no. 7082, 297–302.
- [22] Paul W. K. Rothmund and Erik Winfree, *The program-size complexity of self-assembled squares (extended abstract)*, STOC '00: Proceedings of the thirty-second annual ACM Symposium on Theory of Computing (Portland, Oregon, United States), ACM, 2000, pp. 459–468.
- [23] Rebecca Schulman and Erik Winfree, *Programmable control of nucleation for algorithmic self-assembly*, DNA: International Workshop on DNA-Based Computers, LNCS, 2004.
- [24] ———, *Synthesis of crystals with a programmable kinetic barrier to nucleation*, Proceedings of the National Academy of Sciences **104** (2007), no. 39, 15236–15241.
- [25] David Soloveichik, Matthew Cook, and Erik Winfree, *Combining self-healing and proofreading in self-assembly*, Natural Computing **7** (2008), no. 2, 203–218.
- [26] David Soloveichik and Erik Winfree, *Complexity of self-assembled shapes*, SIAM Journal on Computing **36** (2007), no. 6, 1544–1569.
- [27] Erik Winfree, *Algorithmic self-assembly of DNA*, Ph.D. thesis, California Institute of Technology, June 1998.
- [28] ———, *Self-healing tile sets*, Nanotechnology: Science and Computation (Junghuei Chen, Natasa Jonoska, and Grzegorz Rozenberg, eds.), Natural Computing Series, Springer, 2006, pp. 55–78.
- [29] Erik Winfree, Furong Liu, Lisa A. Wenzler, and Nadrian C. Seeman, *Design and self-assembly of two-dimensional DNA crystals.*, Nature **394** (1998), no. 6693, 539–44.

A Formal 2GAM Definition

Hindrance Map To permit a supertile to interweave itself into an attachable position requires modeling translations of a supertile at the resolution of the size of individual units of the tile face geometries. To this end we define a *Hindrance Map* for a supertile α that represents the set of positions that a supertile takes up, including the bodies of each tile in the supertile, along with the positions blocked by each geometry for each tile face.

Formally, consider a supertile α . The Hindrance Map H_α is the following set of positions: For each tiled position (x, y) in supertile α , the following points are defined to be in H_α . (Please refer to Figure 7 for depictions of tile components in terms of w and ℓ , and note that the entire area occupied by a

BODY: $\{(i, j) | x \cdot (w + \ell) + w \leq i < x \cdot (w + \ell) + w + l, \\ y \cdot (w + \ell) + w \leq j < y \cdot (w + \ell) + w + l\}$

WEST GEOMETRY: $\{(i, j) | x \cdot (w + \ell) \leq i < x \cdot (w + \ell) + w, \\ y \cdot (w + \ell) + w \leq j < y \cdot (w + \ell) + w + l, \\ (i - (x \cdot (w + \ell)), j - (y \cdot (w + \ell) + w)) \in \text{WestGeometry}(\alpha(x, y))\}$

SOUTH GEOMETRY: $\{(i, j) | x \cdot (w + \ell) + w \leq i < x \cdot (w + \ell) + w + l, \\ y \cdot (w + \ell) \leq j < y \cdot (w + \ell) + w, \\ (i - (x \cdot (w + \ell) + w), j - (y \cdot (w + \ell))) \in \text{SouthGeometry}(\alpha(x, y))\}$

NORTH GEOMETRY: $\{(i, j) | x \cdot (w + \ell) + w \leq i < x \cdot (w + \ell) + w + l, \\ (y + 1) \cdot (w + \ell) \leq j < (y + 1) \cdot (w + \ell) + w, \\ (i - (x \cdot (w + \ell) + w), j - ((y + 1) \cdot (w + \ell))) \in \text{NorthGeometry}(\alpha(x, y))\}$

EAST GEOMETRY: $\{(i, j) | (x + 1) \cdot (w + \ell) \leq i < (x + 1) \cdot (w + \ell) + w, \\ y \cdot (w + \ell) + w \leq j < y \cdot (w + \ell) + w + l, \\ (i - ((x + 1) \cdot (w + \ell)), j - (y \cdot (w + \ell) + w)) \in \text{EastGeometry}(\alpha(x, y))\}$

The final hindrance map H_α is the union of the sets BODY, SOUTH, NORTH, EAST, and WEST for each tiled position of α .

Planar Translation of Supertiles Given two supertiles α and β , a *collision free* translation of β with respect to α is any translation of H_α that can be obtained by a sequence of unit translations $\{v_0, u_1, u_2, u_3, \dots, u_r\}$ where v_0 is an initial translation that shifts H_α such that all positions of H_α are northwest of all positions of H_β , and each u_i is one of the translations $\{u_n = (0, 1), u_e = (-1, 0), u_s = (0, -1), u_w = (-1, 0)\}$. Further, after each translation u_i , it must be the case that H_α does not overlap H_β . A *grid locked collision free* translation is a collision free translation in which H_α has been shifted by multiples of $w + \ell$ in both the x and y direction. We are interested in grid locked translations as they correspond to direct translations of α at the resolution of tiles, rather than the higher resolution translations of H_α . We require grid locked translations for a supertile to attach to another supertile as such a translation is needed for the tiles to *line up*.

2GAM Model A Two-Handed Planar Geometric Tile Assembly System consists of a duple (T, τ) where T is a set of geometric tile types and τ is the positive integer temperature of the system. Given a 2GAM system $\Gamma = (T, \tau)$, a supertile is producible if either it is a single tile from T , or it is the τ -stable result of a grid locked collision free translation of two producible assemblies. A supertile α is terminal if for every producible supertile β , α and β cannot be τ -stably attached. A 2GAM is directed (e.g., deterministic, confluent) if it has only one terminal, producible supertile. Given a connected shape $X \subseteq \mathbb{Z}^2$, a 2GAM Γ produces X uniquely if every producible, terminal supertile places tiles only on positions in X (appropriately translated if necessary).

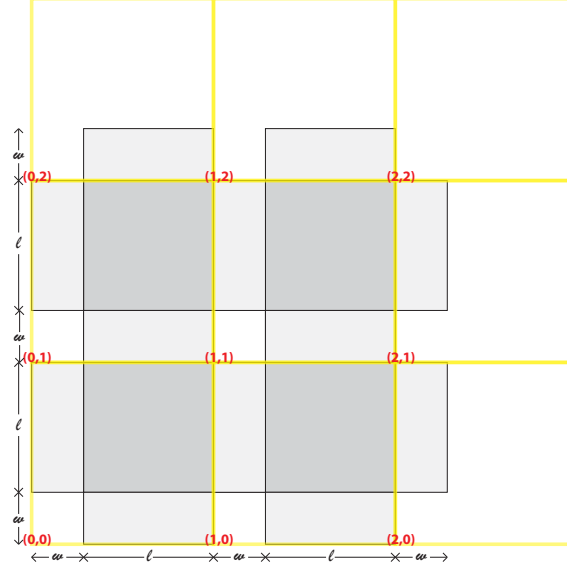
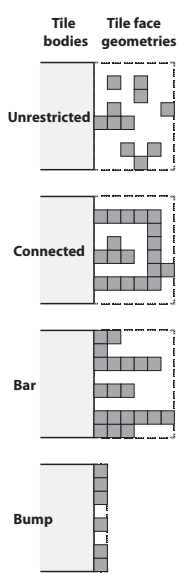


Figure 7: Mapping of tile coordinates (red) to dimensions and units of tile bodies and geometries.

B Classes of Tile Face Geometries

Here we discuss different classes into which tile face geometries can be classified. Note that all GTAM results presented in this paper have “bump” geometries, while the 2GAM result (in its 2-dimensional form) has “unrestricted” geometries. See Figure 8 for an example of each class.



Unrestricted This is the most general class of geometries and places no restrictions on the portions of a tile face geometry region (i.e. the $w \times \ell$ rectangle) which are filled-in and which are empty. Such geometries may be infeasible to implement as the pieces of a geometric face may not be connected to the tile body. However, given a third dimension it is plausible that such a scheme might be implemented by attaching particles to the face of a substrate which also attaches to the tile body.

Connected A slightly more restricted class, the connected class allows arbitrary patterns to be filled-in within the tile face geometries as long as all such portions retain a connected path to the tile body.

Bar A bar geometry is restricted to lines of filled-in points which are connected to the tile body and extend directly away from it. Each bar can be of length x where $0 \leq x < w$.

Bump Bump geometries are the simplest possible types of geometry and consist of a set of points which are directly connected to the tile body. This class can be thought of as simplified bar geometry with $w = 1$.

Figure 8: Classes of tile geometries.

C Additional Details for the $O(\sqrt{\log n})$ GTAM Square Construction

C.0.1 Construction Notation

Consider a positive integer n (the width of the square we wish to assemble). For the sake of clarity, assume n is even.

- Let $n' = \lceil \log \frac{n+1}{2} \rceil$.
- Let $r = \lceil \sqrt{(1/2)n' + 1/2} \rceil$.
- Let $B = 2^{n'} - n/2 - 1$.

C.0.2 Base $O(\sqrt{\log n})$ Counter at Temperature $\tau = 1$

The first step of the construction is the assembly of a $2 \times \log n$ rectangle that will serve as a bed for a third layer that will place a row of tiles that represent a $\log n$ bit binary number. The $\tau = 1$ GTAM system used for this portion of the assembly is obtained by applying the transformation from Theorem 3.6 to convert an efficient $\tau = 2$ ATAM system into a equivalent $\tau = 1$ GTAM system.

The temperature $\tau = 2$ ATAM system that will be converted is described in Figure 10 and constitutes a 2-digit, base r counter. The counter works within the ATAM at temperature $\tau = 2$ and is a generalization of the base-2 version first described in [22]. By specifying the east glues of tiles S_1 and S_2 , the counter can be *seeded* to any specified starting value. In the tile set given in the figure, the counter is initialized to value 0. The value of the counter is incremented at every other column as the assembly grows from west to east, finally halting when the counter rolls over to 0. Thus, for a given choice of r and an initial seed value b , the final assembly will be a $2 \times 2r^2 - 2b$ rectangle. For our construction, we utilize $r = \lceil \sqrt{(1/2)n' + 1/2} \rceil$, which guarantees enough room to place a length n' binary string in the next step of the construction. As we can initialize the counter to be shorter if needed, we assume the counter has been initialized to grow to length exactly $n' + 2$ (an extra 2 positions are not used to encode bits in our constructions, thus the extra 2 length).

To modify the ATAM system of Figure 10 to a $\tau = 1$ system, we observe that it is a zig-zag system according to Definition 3.4 (rotated 90 degrees). Therefore, we can apply Theorem 3.6 to obtain an equivalent $\tau = 1$ GTAM system shown in Figure 11. The general case details of the conversion are detailed in Section 3.2, but the basic idea of the transformation is to replace east strength 1 glues with the null glue type, and assign a unique geometry to each edge for each glue type. In particular, for an east/west glue type x , a corresponding pair of geometries are computed, $E(x)$ and $W(x)$, such that $E(x)$ is incompatible with all other geometries within the system with the exception of $W(x)$, and vice versa. Each occurrence of the glue type x on the east face of a tile type is replaced by a GTAM tile type with geometry $E(x)$ for the east face geometry. The same replacement by $W(x)$ is done for west occurrences of x . This geometry assignment is also applied to all north/south glue types as well. The result is a system that assembles in the same fashion as the original temperature $\tau = 2$ system and with the same tile complexity, but does so at temperature $\tau = 1$.

C.0.3 Bit Decoder Tiles

The next step in the construction consists of a collection of decoder tiles which grow across the surface of the $2 \times 2r^2$ rectangle assembled from the previous step. The general tile set for these decoder tiles is given in Figure 4. The growth of these tiles is initialized by the blue glue displayed by the final tile placement of the base r counter from the previous section. The decoder tiles consist of a repeating chain of $2r$ tiles with labels 0 to $2r - 1$. The west geometry of tiles with label i are compatible with the east geometries of tiles with label $(i + 1) \bmod 2r$, and incompatible with all other geometries. Thus, the chain of tiles must assemble in the proper order. Further, there exactly 2 tile types with each integer label from 0 to $2r - 1$, a *green* type and an *orange* type. Our goal is to assemble a supertile that encodes a given target binary string

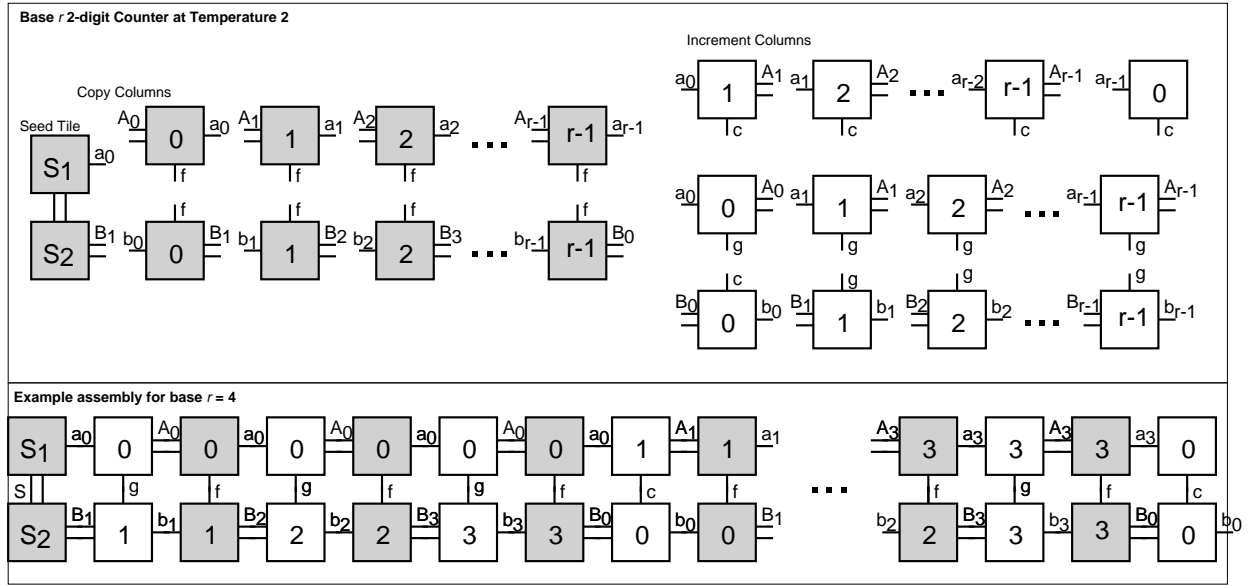
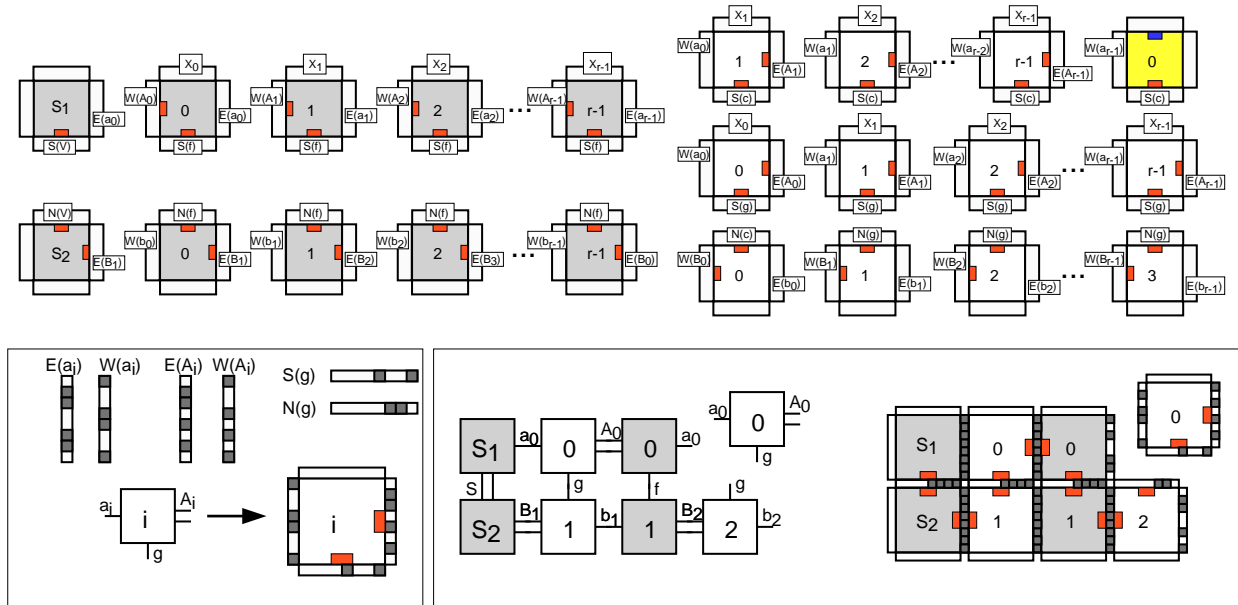


Figure 10: This figure contains a tile system that assembles a $2 \times r^2$ rectangle for a given integer r . The construction is an implementation of a 2-digit, base- r counter with tile complexity $5r + 2$ that assembles at temperature $\tau = 2$ in the standard ATAM.



along its surface. The encoding is the pattern of green and orange tiles with orange tiles representing binary 0 bits, and green representing binary 1 bits. The key to get the goal binary string assembled is to enforce that at each bit position the correct bit is chosen. We do this by appropriately assigning geometry to the south face of the decoder tiles and the north face of the base r counter tiles.

In more detail, suppose we are given a target binary string $B = b_{2r-1} \dots b_2 b_1 b_0$ to be assembled, the geometries G_j , \bar{G}_j , and X_i are assigned such that G_j and X_i are compatible if and only if $b_{2r(r-1)-2ri+j} = 0$. A detailed example of such a compatibility matrix is given in Figure 4, along with a sample set of geometry assignments to tile faces that satisfies such constraints. More generally, such compatibility constraints can be achieved with geometry of length r for a length $2r^2 - 2$ string B by way of Theorem G.5, which is asymptotically the best achievable in most cases. In the case of less complex binary strings, more compact geometries can be obtained as discussed in Section G. From these compatibility constraints, the desired target binary string is guaranteed to assemble. Further, the final placed tile can be specified as a special type with a north *purple* glue which seeds the next portion of the construction.

C.0.4 Binary Counter Tiles

The next portion of the construction, seeded by the final tile placed during the decoder tile portion, consists of a binary counter tile system which grows north, start from the binary string decoded in the previous section, up until the counter rolls over. The construction is a $\tau = 1$ GTAM version of a well known temperature $\tau = 2$ ATAM construction of $O(1)$ tile types [22]. The counter increments every other row, and thus will grow to a height of exactly $2^{n'+1} - 2B - 2$, where B is the initial value of the counter which is the binary string encoded in the previous phase of the construction. Our goal is for the counter to build to a height equal to n , the dimension of the goal $n \times n$ square. We thus choose B in the previous section to be $B = 2^{n'} - n/2 - 1$.

Finally, the construction is finished by observing that the binary counter construction can be implemented such that the final tile placed at the northeast corner of the assembly exposes a glue type which seeds the assembly of a rectangle that grows east for exactly $n - n' - 2$ units. This can be accomplished in the exact same way we achieved a north growing rectangle of length n , but with an alternate choice of initial binary string assignment. This construction can in turn seed a south growing rectangle of the same length. This final rectangle can seed the growth of a final $O(1)$ size collection of tile types which fills in the casing of the hollow $n \times n$ square, yielding the final full $n \times n$ square. A high level figure depicting the construction is given in Figure 9.

D Details of Zig-Zag Simulation

Definition D.1. Nice Zig-Zag System. A zig-zag system $\Gamma = (T, \tau, s)$ is called a *nice* zig-zag system if:

1. The terminal assembly of Γ contains no exposed east-west non- glue types.
2. All producible assemblies of Γ contain no mismatched glues.

Definition D.2. Tile System Simulation. An ATAM or GTAM system $\Gamma_2 = (T_2, \tau_2, s_2)$ is said to simulate a second ATAM or GTAM system $\Gamma_1 = (T_1, \tau_1, s_1)$ if:

1. There exists a function $F : T_1 \rightarrow P(T_2)$ such that an assembly sequence $\langle (t_1, x_1, y_1), \dots (t_i, x_i, y_i) \rangle$ is valid for system Γ_1 if and only if there exists tile types $r_i \in F(t_i)$ such that the assembly sequence $\langle (r_1 \in F(t_1), x_1, y_1), \dots (r_i \in F(t_i), x_i, y_i) \rangle$ is valid for Γ_2 .

The tile complexity scale factor of the simulation is defined to be $|T_2|/|T_1|$. This definition only considers the case of simulating a system without scaling the size of the assembly (scale factor 1). See [9] for a more general definition of simulation that permits scaled assembly size factors.

Observation D.3. The “simulate” relation between tile systems is transitive. Further, if system A simulates B with tile type scale factor x , and B simulates C with tile type scale factor y , then A simulates C with tile type scale factor xy .

Lemma D.4. Any zig-zag system $\Gamma_1 = (T_1, \tau_1, s_1)$ can be simulated by a nice zig-zag system $\Gamma_2 = (T_2, \tau_2, s_2)$ with tile type scale factor $|T_2|/|T_1| = O(1)$.

Proof. of Theorem 3.5: Consider a zig-zag system $\Gamma = (T, 2, s)$. By Lemma D.4, there exists a zig-zag system $\Gamma' = (T', 2, s')$ that simulates Γ with $O(1)$ tile type scale such that the assembly of Γ' has the “nice” properties described in Definition D.1.

We now define a system $\Upsilon = (R, 1, q)$ that simulates Γ' with $O(1)$ tile type scale, and thus simulates Γ with $O(1)$ tile type scale by the transitivity of simulation.

Let σ'_w denote the set of all west-east glues that are represented in the tile set T' . Let σ'_n denote the set of all north-south glues that are represented in the tile set T' . Let H denote an injective mapping from the glue types represented in T' to some new set of strength-1 glues ρ .

For each $t \in T'$, define the geometric tile type r_t as follows. Denote the north, south, east, and west glues of t as $north(t)$, $south(t)$, $east(t)$, and $west(t)$ respectively. Let the west glue of r_t be $west(r_t) = H(west(t))$ and $east(r_t) = H(east(t))$. If $north(t)$ is a strength-2 glue, then $north(r_t) = H(north(t))$, otherwise $north(r_t) = null$. If $south(t)$ is a strength-2 glue, then $south(r_t) = H(south(t))$, otherwise $south(r_t) = null$.

We assign the empty set geometry to all east/west edges of tile types in R . We assign non-empty geometries to north/south edges of each tile type r_t such that the south edge of a given $r_t \in R$ is compatible with a north edge of a tile type $r_v \in R$ if and only if $south(t) = north(v)$. By Theorem G.9, we can achieve such compatibility requirements with a geometry of size at most $\log \sigma'_n + \log \log \sigma'_n$.

We now show that the system $\Upsilon = (R, 1, r_{s'})$ simulates $\Gamma' = (T', 2, s')$ (with tile type scale factor 1). Consider the function $F(t) = \{r_t\}$ to map tile types from T' to R . Suppose Υ correctly simulates Γ' up to the first $i-1$ steps. That is, for the first $i-1$ assembly sequence steps $\langle (t_1, x_1, y_1), \dots, (t_{i-1}, x_{i-1}, y_{i-1}) \rangle$ of the unique assembly sequence of Γ' , the first $i-1$ steps of the assembly sequence of Υ are $\langle (r_{t_1}, x_1, y_1), \dots, (r_{t_{i-1}}, x_{i-1}, y_{i-1}) \rangle$. Consider the i^{th} step in the assembly sequence of Γ' , (t_i, x_i, y_i) . We know that the placement position (x_i, y_i) must occur north, west, or east of the previously placed tile position (x_{i-1}, y_{i-1}) . If the placement occurs to the north, then (r_{t_i}, x_i, y_i) is a valid step in the assembly sequence for Υ . This is because the tile transformation explicitly assigns tile type $r_{t_{i-1}}$ a strength-1 north glue that matches the south glue of r_{t_i} , as t_{i-1} must have a north strength-2 glue that matches the south glue of t_i . Further, by the fact that Γ' is nice, there are no exposed east/west glue faces of the Γ' assembly after $i-1$ steps, and thus the Υ assembly will have no other exposed glues (of strength-1) beyond the single north glue of $r_{t_{i-1}}$. This implies that the placement of r_{t_i} at position (x_i, y_i) is the only possible next tile placed for system Υ .

Now suppose the i^{th} tile attachment occurs to the east of the previously placed tile in the Γ' assembly. The required strength-2 attachment threshold for the i^{th} tile can be achieved by two strength-1 glues, or by a single strength-2 glue. In the cooperative strength-1 glue case, we know that r_{t_i} is the only tile in R that both matches the east glue of $r_{t_{i-1}}$ and has a compatible geometry with the north face of the tile type at position $(x_i, y_i - 1)$. Thus, (r_{t_i}, x_i, y_i) is a valid next element of Υ 's assembly sequence, and is the only valid next element by the nice properties of Γ' . For the case of a strength-2 east attachment, we know that the (r_{t_i}, x_i, y_i) attachment is valid for Υ because of the matching strength-1 east glue of $r_{t_{i-1}}$ and west glue of r_{t_i} , and because the tile type south of (x_i, y_i) , if there is one, is guaranteed to have a compatible north geometry with r_{t_i} by the “no mismatched glue” property of nice zig-zag systems. The attachment is also unique because of the “no exposed glues” property of nice zig-zag systems. The remaining west attachment case is analogous to the east attachment case.

Therefore, $\Upsilon = (R, 1, r_{s'})$ simulates $\Gamma' = (T', 2, s')$ with tile type scale $|R|/|T'| = 1$, and therefore also simulates $\Gamma = (T, 2, s)$ with tile type scale $O(1)$. The size of the geometry of the simulation is at most $\log \sigma'_n + \log \log \sigma'_n$ by Theorem G.9, and is thus $\log 2\sigma_n + \log \log 2\sigma_n = \log \sigma_n + \log \log \sigma_n + O(1)$ where σ_n is the number of north/south glue types in T . \square

Proof. of Theorem 3.6: We use the same approach for simulating nice zig-zag systems as given in the proof for Theorem 3.5, with the modification that all non-null glue types from system T' are mapped to a single strength-1 glue type x . Further, rather than empty set geometry assigned to all east west glues in the simulation set, we assign geometries that satisfy a compatibility matrix which assigns a 0 to entries which

correspond to identical glues, and 1 to entries corresponding to non-identical glues. Thus, while there is a single glue, only the appropriate tile with geometry representing the appropriate east/west glue will attach. By the same analysis given for Theorem 3.6, we achieve a simulation set for any zig-zag system with $O(1)$ tile type scale and $\log \sigma + \log \log \sigma + O(1)$ geometry size. \square

E Additional details for 2GAM Results

This section includes additional details and images describing the construction in Section 4.

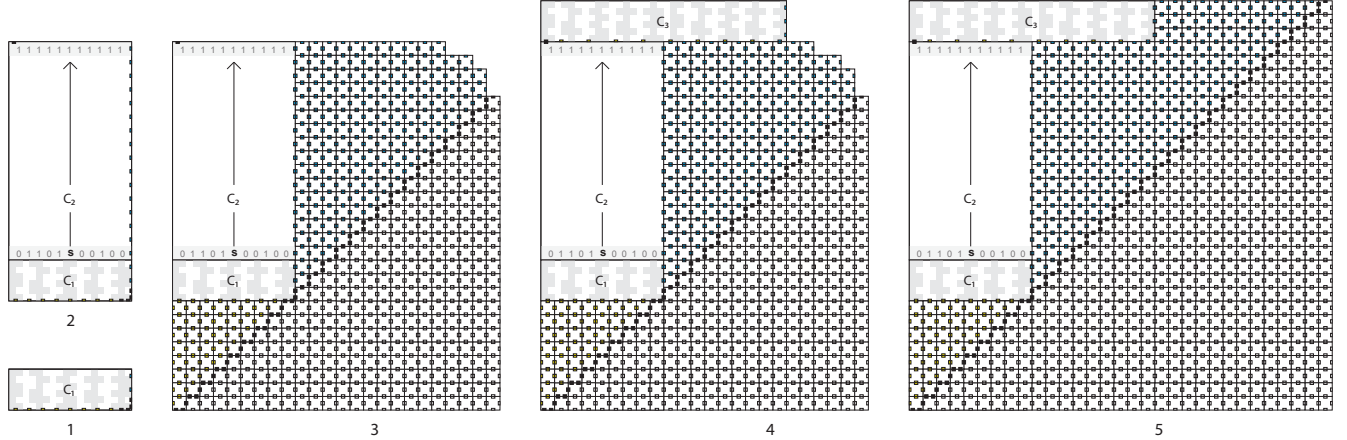


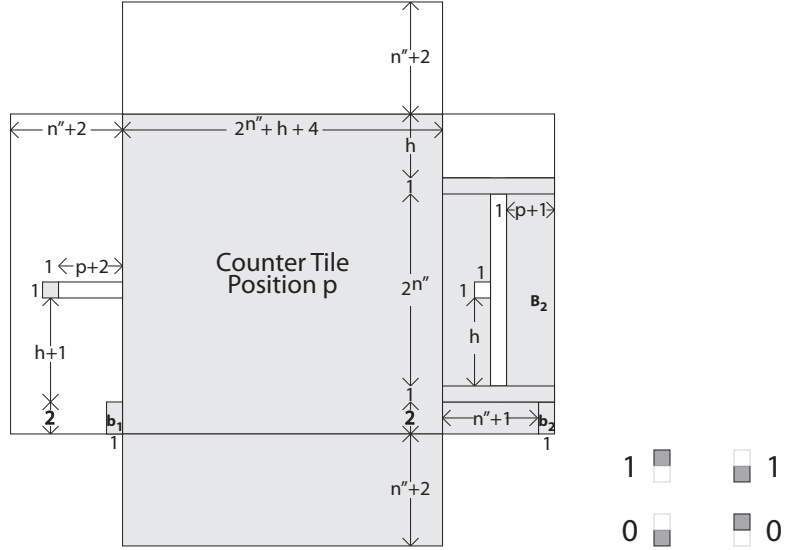
Figure 12: A sketch of the assembly sequence of the square and its components: 1. first the counter module C_1 assembles; 2. the initial value s is seeded by C_1 , which allows C_2 to assemble; 3. filler tiles complete a sufficient portion of the square; 4. A fully formed version of C_3 attaches; and 5. filler tiles complete the square.

E.1 The 2-handed counters C_1 and C_3

For each tile in this construction, the bodies are squares of $(2^{n''} + h + 4) \times (2^{n''} + h + 4)$ geometric units. (See Figure 13a for an example.) The north and south geometries consist of $(2^{n''} + h + 4) \times (n'' + 2)$ rectangles of geometric units (i.e. rectangles of length $\ell = 2^{n''} + h + 4$ and width $w = n'' + 2$). All north geometries are completely empty, while all south geometries are completely filled in. However, the east and west geometries are composed of $(n'' + 2) \times (2^{n''} + h + 4)$ rectangles of geometric units (i.e. rectangles of length $\ell = 2^{n''} + h + 4$ and width $w = n'' + 2$) which contain intricate collections of gaps, which we call *sockets*, and projections, which we call *prongs*.

Note that while the techniques utilized in this construction can be generalized to form counters of arbitrary bases, the tile complexities are asymptotically identical regardless of the base, so for simplicity of explanation we utilize only base 2 counters.

The purpose of the counter module C_1 is to count from 0 through $2^{n''} - 1$, for a total of $2^{n''}$ values, each represented by exactly one column. The northern glue on the northernmost tile of each of column is used to represent one of the bits of the value s (from left to right, the most significant bit to the least significant). Each column is composed of $n'' + 4$ tiles arranged vertically. The tiles in each column perform two tasks: 1. represent an n'' -bit binary number m for $0 \leq m < 2^{n''} - 1$, and 2. present a northern glue which represents the correct value (0 or 1) for the $2^{n''} - (m + 1)$ th bit of s . The southern n'' tiles, which we call the *counter* tiles, each represent one bit of m , from top to bottom the most significant bit to the least significant. The northernmost 4 tiles are gadgets called *caps* which serve to represent the bit values on the north of the columns, while ensuring their correct positioning relative to the bits of s . There are two possible caps which can form, a 0-cap and a 1-cap. Either cap can nondeterministically attach to any column of counter tiles to



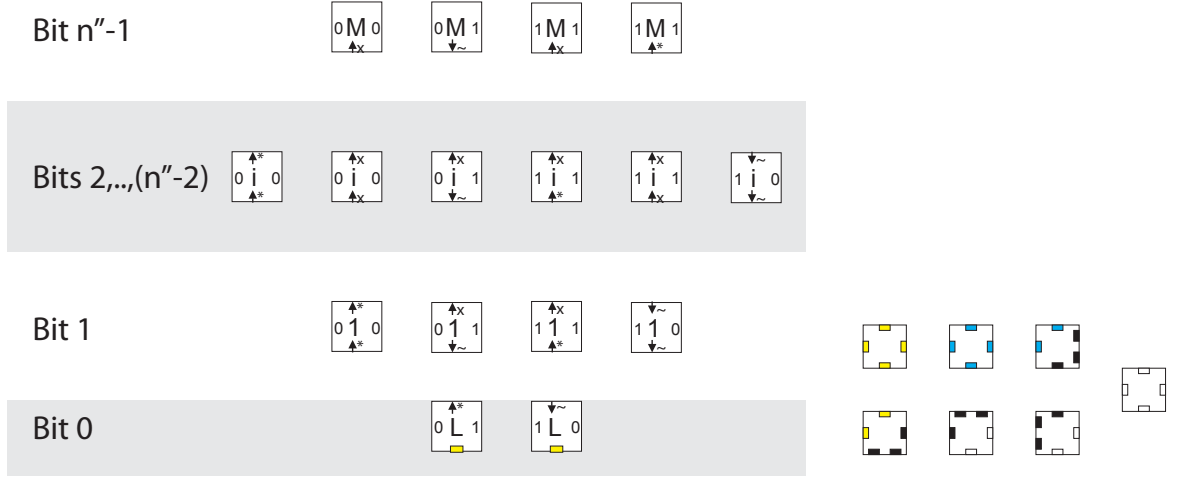
(a) Dimensions of the geometric units of a counter tile for bit position p , representing bit b_1 on its west side and b_2 on its east side. White portions indicate areas which are not filled in, while grey areas are filled in (although areas b_1 , b_2 , and B_2 contain a mixture).

(b) Patterns used to represent bit values b_1 (left) and b_2 (right) on the southern corners of west and east geometries, respectively.

Figure 13: Details of the geometries of the counter tiles.

provide the 4 northernmost tiles of any column (except for the leftmost and rightmost columns, which are special cases discussed later). This allows for the formation of two versions of the columns which represent each counter value m : one that represents a 0 on the north and one that represents a 1. It is the purpose of the cap to ensure that only the version of the column with the correct bit of s (i.e. for the column representing the value m and thus the $2^{n''} - (m + 1)$ th bit of s , the one representing the bit value matching the $2^{n''} - (m + 1)$ th bit of s) can attach to neighboring columns.

Each column can completely form independently of every other column, and in fact must be fully formed before it can combine with any other column. For each bit position of the counter, there are two tile types, one representing a 0 and one a 1. Only those for the least significant bit position have strength-1 glues on their east and west, while those for every other bit position have *null* glues on their east and west sides. However, all counter tiles have geometries on their east and west sides which will be explained shortly. The north and south sides of these tiles have flat geometries and strength-2 glues which allow them to nondeterministically combine with the bit values above and below (but only in the correct order of significance). Thus, the counter portions of each column can form by nondeterministic combinations of exactly one tile type for each bit position, and therefore columns can form which represent each of the values 0 through $2^{n''} - 1$. For each counter value, two columns can form: one with a 0-cap and one with a 1-cap. The east and west sides of the cap tiles have geometries but *null* glues, except for the northernmost which has strength-1 glues on the east and west. The north and south sides of each cap tile are similar to those of the counter tiles in that they have only flat geometries and strength-2 glues, except for the northernmost cap tile which has a strength-1 glue that represents the bit of s provided by that column. (The rightmost, and therefore least significant, bit of s will be represented by a strength-2 glue, and both the most significant and least significant bits will be represented by special glues which convey both the bit values and the most and least significance of those positions.) The positioning of the two strength-1 east and west glues in each column, on only the top and bottom tiles, ensures that the only way that two columns can possibly combine with each other (keeping in



(a) Template for the tile set which allows the formation of columns representing binary numbers in the range 1 through $2^{n''} - 2$ (preventing the formation of numbers 0 and $2^{n''} - 1$), and which increment the binary numbers represented by each column from west to east. The bit significance positions for each group of tiles are shown. Note that a unique group of tile types is created for every bit position, although those for positions 2 through $n'' - 2$ are shown in a single group. Each tile has strength-2 glues on its north and south edges (except for the north edge of the northernmost group and the south edge of the southernmost group). The east and west bit values are represented in the actual tile set by the east and west geometries.

(b) Filler tiles which fill in the bulk of the square.

mind that this construction operates at temperature 2) is if they are fully formed (i.e. they contain all 4 cap tiles and exactly one counter tile for each bit position).

E.1.1 Counter tiles

Figure 14a shows a tile set which performs portions of the functionality of the counter tile types. This tile set is designed to form $1 \times n''$ tile columns representing n'' -bit binary numbers, with specific groups of tile types designed for various groups of bit positions. All tiles have strength-2 glues on their north and south sides (except for the north glue of those in the most significant bit position and the south glue of those in the least significant bit position) which match only those of the adjacent edges of tiles for adjacent bit positions. The columns formed represent n'' -bit binary numbers on their west and east sides such that for each number x represented on the west, the value $x + 1$ is represented on the east. Additionally, every possible n'' -bit binary value can form *except* for the values 0 and $2^{n''} - 1$ (i.e. all 0's or all 1's). Those values are represented by the leftmost and rightmost columns, respectively, of the counter C_1 and are formed by a separate set of hard coded tile types since the leftmost column must expose a special glue on the north which marks that bit of s as the most significant bit, and the rightmost column must expose a special strength-2 glue on the north which marks that bit of s as the least significant bit, and it must also have flat geometries and strength-1 glues along its eastern side which can attach to the filler tiles.

For each bit position p (where $0 \leq p < n''$), in order to represent bit values b_1 on the west and b_2 on the east, the geometries are designed as shown in Figure 13a. The 1×2 rectangles labeled b_1 and b_2 each have exactly one position filled in and one left empty, depending on the bit values being represented and following the patterns shown in Figure 13b. It is these regions which enforce the restriction that only if two columns represent the same bits in all positions can they line up and come together completely to combine. The remaining area of the west geometry is completely empty except for one unit located $h + 3$ units from the southern edge and $p + 2$ units from the east. This single point is referred to as the prong, which will slide into a complementary socket in the east geometry of a tile in the same bit position of a column to the left.

The portion of the east geometry to the left of region b_2 is a $2 \times n'' + 1$ rectangle which is completely filled in. The geometry immediately to the north of that creates the socket, which has one empty column $p + 1$ units from the east and one additional empty spot h units up that column and one unit to the west. The area labeled B_2 consists of $p + 1$ columns which are divided horizontally into a number of blocks dependent upon p , the bit position being represented. It is divided into $2^{n''-p}$ blocks of width $p + 1$ and height 2^p . Those blocks can be thought of as being numbered from the southernmost as 0, through the northernmost as $2^{n''-p}$. If $b_2 = 0$, then each even numbered block is empty and each odd numbered block is filled in. If $b_2 = 1$, then the even numbered blocks are empty and the odd numbered blocks are filled in. Examples can be seen in Figure 17.

Intuitively, the eastern socket is split into two groups of regions - those representing a 0 and those representing a 1. If the socket is part of a 0-tile (i.e. a tile which represents a 0 for the particular bit of the counter value), the regions representing 0 are left open and those representing 1 are filled in. For a 1-tile, the opposite is true. The number of regions is determined by the bit position. For the most significant bit position, there are only two regions, with the bottom representing 0 and the top 1. The tile for the next bit down has four regions, which are essentially each of the regions of the previous tile split into 2 sections, with the bottom of each new pair representing 0 and the top 1. This splitting and doubling continues downward until the tile for the least significant bit which has $2^{n''}$ regions consisting of individual squares. In a 0-tile, the bottom of each pair starting from the bottom is left open and the top of each is filled in, and vice versa for a 1-tile.

The portion of the east geometry above the socket, which is an empty $n'' + 2 \times h$ rectangle, is referred to as the *padding* region. A padding region exists on the north of each west and east geometry, and its purpose is to guarantee that the geometries of tiles in one bit position of a column will not collide with the geometries of those in adjacent bit positions of a column correctly combining with it. This padding region is sufficient because the maximum relative translation that two columns which can combine with each other will ever need to experience is a vertical offset of $2^{n''}/2$, and the padding region plus the separation provided by the north-south geometries of combined tiles is at least that large.

E.1.2 Cap tiles

The remainder of each column of C_1 consists of four tiles called the *cap* which combine to form a vertical column one tile wide and 4 tiles high. There are two possible types of cap, a 0-cap and a 1-cap, which can nondeterministically attach to any partial column representing a counter value to form the top four tiles of a complete column. It is the job of a b -cap (for $b = 0$ or 1) to ensure that only if it has attached to a counter value y (which will be positioned at the location of the $(2^{n''} - (y + 1))$ th bit of s) and b matches that bit of s , only then can that column attach to other columns within C_1 (specifically, those columns representing $y - 1$ and $y + 1$) and thus become a portion of that counter. Otherwise, an unmatching column (i.e. one to which the “wrong” cap has nondeterministically attached) is relegated to use within C_3 , which will be described later.

Please refer to Figure 14 for a graphical depiction of the geometries of the cap tiles. The north and south geometries of all cap tiles are the same as those of the counter tiles: a completely filled in $(2^{n''} + h + 4) \times (n'' + 2)$ south geometry and a completely empty $(2^{n''} + h + 4) \times (n'' + 2)$ north geometry. The prong in the west geometry of cap tile 0 is located $h + 3$ units from the south and the prong in east geometry of cap tile 1 is located $2h + 3$ units from the north. Each consists of a single filled-in location 3 units away from the bodies. The remainder of those geometries are empty. Additionally, the west geometry of cap tile 2 and the east geometry of cap tile 3 are completely empty. Thus, it is the 4 sockets, one on each cap tile, which provide the bulk of the functionality of the caps.

The east geometries of cap tiles 0 and 2 each have two empty rows on the south and a buffer region on the north. In between are sockets which each have one completely filled-in row above and below them. The west geometries of cap tiles 1 and 3 each have one filled-in row on the south, a buffer region on the north, two empty rows south of the buffer, one filled-in row south of that, and sockets in between the two filled-in rows. The socket patterns of the cap tiles occupy a single column and are referred to as S_1 and S_2 , plus their complements $\overline{S_1}$ and $\overline{S_2}$. (Note that the sockets $\overline{S_1}$ and $\overline{S_2}$ are not “utilized” during the combination

of columns which form into C_1 , but only for those which become part of C_3 .) These columns are $2^{n''}$ units tall, and by associating each of these units with one of the $2^{n''}$ possible values of a counter column to which a cap can attach, it is possible to utilize the pattern of gaps and blocks in a socket to determine which type of cap (0 or 1) can be attached to a particular counter value while allowing that column to become part of C_1 . By virtue of the fact that the prongs are relatively short compared with those of the counter tiles, and due to the patterns of sockets and prongs on the counter tiles, compatible columns (which can only combine if they are completely formed, so we only consider completely formed columns in this discussion) are forced to align in such a way that, during their combination, the prong of a cap which is incident upon the socket of a neighboring cap will be at a position which corresponds to the value represented by the counter value. In this way, we can allow the prong to slide into an “accepting” gap if and only if the cap type is correct for that value, namely if that counter value should be associated with a 0 or 1 cap and thus a 0 or 1 bit of the number s .

To provide this functionality, the socket patterns for a cap of type b are the following. S_1 treats the $2^{n''}$ units as representing each value k for $0 \leq k < 2^{n''} - 1$ counting downward from the top to bottom. For the k th unit, if k coincides with a numbered bit position of s which has the bit value b , there is a gap, otherwise it is filled in. $\overline{S_1}$ is the exact complement (which is equivalent to designing S_1 for bit value $1 - b$). S_2 treats the $2^{n''}$ units as representing one “invalid” value plus each value k for $0 \leq k < 2^{n''} - 2$, counting upward from the bottom to top with the first position being the invalid, and thus always filled-in, position. Again, for the k th unit, if k coincides with a numbered bit position of s which has the bit value b , there is a gap, otherwise it is filled in, and again, $\overline{S_2}$ is the exact complement (with the exception that for both the invalid location is filled in).

Intuitively, the reason for the north-south reversal and offset of 1 for socket values between the pairs of sockets S_1 and S_2 is that the prongs that will be validating the combination of counter-to-cap values will be positioned (during the combination of a pair of columns) relative to a single counter value, and thus the sockets need to use that one value to validate the two columns which represent (in the correct case) two consecutive values. Additionally, the motion of the prongs relative to the respective sockets will be vertically reversed due to the fact that they are on opposite sides of the combining columns. (Also, recall that whenever two columns attempt to combine which do not encode consecutive values, the patterns encoded in regions b_1 and b_2 will prevent their combination.) Essentially, S_2 and $\overline{S_2}$ are designed so that if they are included in a column which encodes the counter value y , the positioning of the prong from a complementary column to the right, which would represent the counter value $y + 1$, would align it with holes if and only if the bit value of the $((y + 1) - 1)$ th, or y th, bit of s equals b .

E.1.3 Buffer columns and C_3

As previously mentioned, every counter column can form in two versions, one with a 0-cap and one with a 1-cap. The design of the glues and geometries is sufficient to guarantee that a fully formed C_1 can and will include exactly one version of each column, namely that which will present the correct value for the corresponding bit of s . In order to ensure that this construction is directed, and thus has exactly one terminal assembly, the versions of columns which are not included in the formation of C_1 must be included somewhere within the eventual $n \times n$ square. That is the purpose of C_3 .

Intuitively, C_3 is a counter of approximately double the length of C_1 which incorporates all of the “bad” columns (meaning those with the wrong caps to be included in C_1), but since they cannot directly combine with each other, each pair is separated by a *buffer* column that not only allows them to combine but ensures that they are in fact bad columns (and hence the approximate doubling of the length as compared to C_1).

Each buffer column is formed of two portions, similar to the other columns. However, the bottom portion of buffer columns simply represent a single binary value from the range 1 through $2^{n''} - 1$, inclusive. This value is represented on both the west and east sides rather than an incremented value appearing on the east side as in the counter columns, and the geometries which represent each bit are the same as those for the counter tiles. The logic provided by the north-south glues must simply ensure that columns representing every binary number from 1 through $2^{n''} - 1$ can form, thus simply excluding a column of all 0’s. (This is because the counter column representing 0 encodes the number 1 on its east, and thus the buffer column

need never combine with a column representing the number 0.) Thus, there will be a buffer column which can be sandwiched by every consecutive pair of bad columns.

There is only one version of each of the buffer cap tiles, which can be seen in Figure 14. Similar to the tiles of the counter columns, the only non-glues on the west and east edges are strength-1 glues on the east and west of the buffer counter tiles in the least significant bit position (the southernmost in each column) and the buffer cap tile 3 (the northernmost in each column). As shown, the only “interesting” geometry is that of the prongs on buffer cap tile 2 and 3. These prongs are positioned so that they must fit into the sockets of complementary counter columns which allow combinations of counter values with cap values that are the opposite of those required for inclusion in C_1 . In this way, full buffer columns can combine on both sides with exactly those full counter columns which received the “wrong” cap and in the correct order to count from 0 through $2^{n''} - 1$ and therefore construct C_3 .

E.2 Details of the counter C_2

Once C_1 has completely formed, the binary number s is encoded on its northern face as a series of strength-1 glues, along with a strength-2 glue on the north of the east-most tile, which is the least significant bit of s . To these glues, tiles of the types shown in Figure 15 attach to perform the counting of s through $2^{n'} - 1$. This is a basic counter which can assemble in both the standard aTAM as well as the 2HAM, resulting in the same produced assembly since it is polyomino-safe (see [28]). It is notable that as each row assembles, it “checks” whether or not it is the maximal value of the counter and, if so, causes a tile to be placed in the west-most position which has a special north-facing glue that will eventually allow C_3 to attach.

E.3 Completion of the square

Figure 12 shows the sequence in which the full assembly can progress. Once C_1 has completely formed and C_2 has completely grown northward from it, filler tiles are able to complete a sufficient portion of the rest of the square that a fully formed (and only a fully formed) C_3 can attach. The filler tiles are shown in Figure 14b. Once C_3 has attached, the remaining filler tiles are able to complete the formation of the full $n \times n$ square.

E.4 An example: constructing a 250×250 square

Here we provide an example of portions of this construction as applied to a 250×250 square.

- n : 250
- n' : $\lceil \log n \rceil = \lceil \log 250 \rceil = 8$
- n'' : $\lceil \log n' \rceil = \lceil \log 8 \rceil = 3$
- s : $2^{n'} + 2^{n''} + 2n'' + 8 - n = 2^8 + 2^3 + 2(3) + 8 - 250 = 28 = 00011100_2$
- h : $2^{n''-1} - 1 = 2^2 - 1 = 3$
- C_1 : 2-handed counter which counts from 0 through $2^{n''} - 1 = 2^3 - 1 = 7$ for a total of $2^3 = 8$ columns
- C_2 : standard counter which counts from $s = 28$ through $2^{n'} - 1 = 2^8 - 1 = 255$ for a total of $2^{n'} - s = 256 - 28 = 228$ columns
- C_3 : 2-handed counter with “buffer” columns which counts from 0 through $2^3 - 1 = 7$ for a total of $2^{n''+1} - 1 = 2^4 - 1 = 15$ columns

Figure 16 shows the dimensions for the components of the square for this example, as well as the dimensions of the tile bodies and geometries of counter tiles.

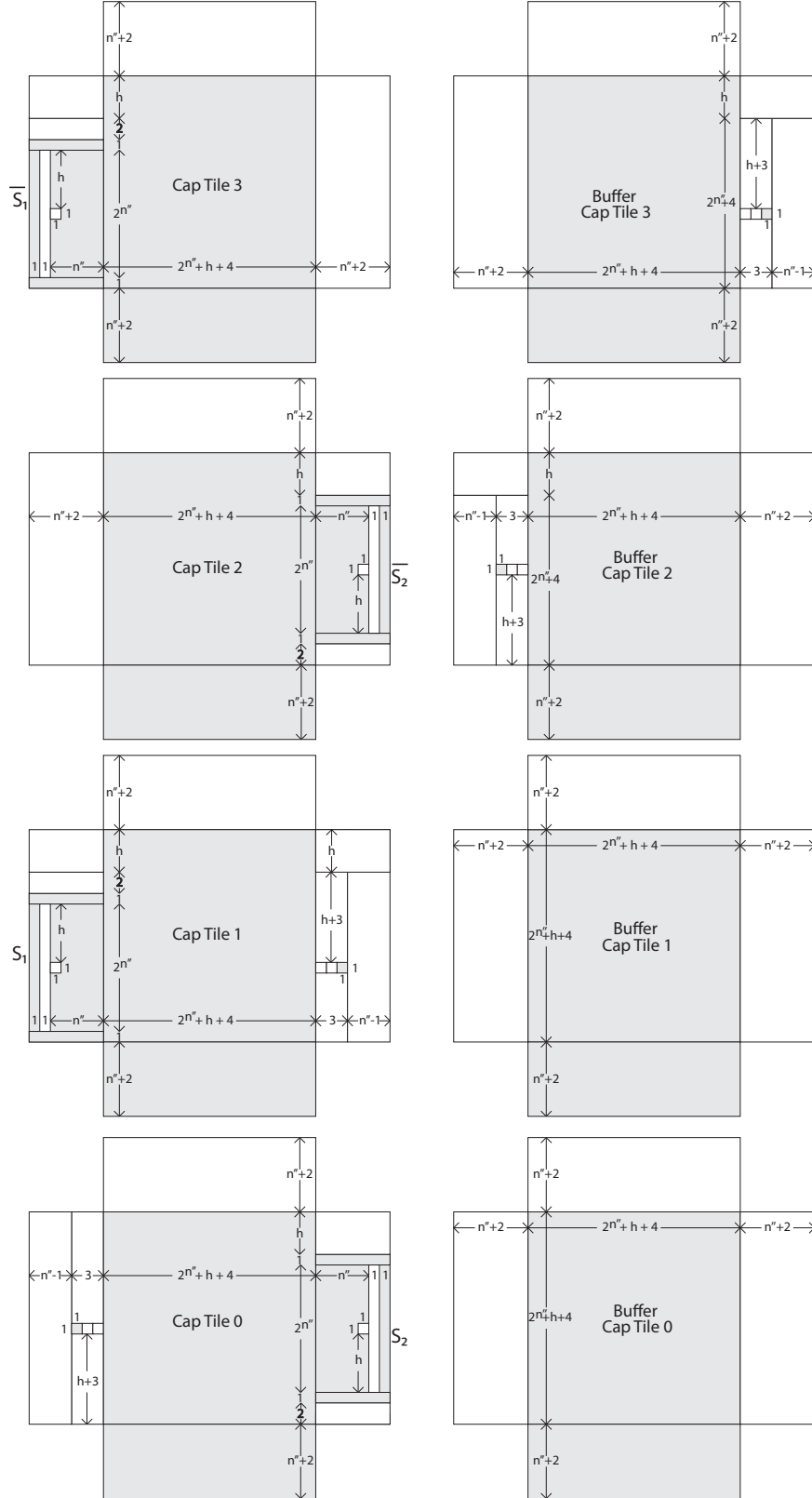


Figure 14: Dimensions of the features in the geometries of the counter column cap tiles as well as the buffer column cap tiles. White portions indicate areas which are not filled in, while grey areas are filled in (although socket areas S_1 , S_2 , \overline{S}_1 , and \overline{S}_2 contain a mixture)

bit position	0	1	2	3	4	5	6	7
bit of s	0	0	0	1	1	1	0	0
S_1 for a 0-cap			X	X	X			
$\overline{S_1}$ for a 0-cap	X	X				X	X	X
S_2 for a 0-cap	X				X	X	X	
$\overline{S_2}$ for a 0-cap	X	X	X	X				X
S_1 for a 1-cap	X	X				X	X	X
$\overline{S_1}$ for a 1-cap			X	X	X			
S_2 for a 1-cap	X	X	X	X				X
$\overline{S_2}$ for a 1-cap	X				X	X	X	

Table 2: Patterns for each combination of socket and cap type. Each "X" represents a location which is filled-in. Note that the numbering for the S_1 sockets starts at the north and increases southward, while it is the opposite for the S_2 sockets.

The geometries of the counter tiles are a straightforward application of the generic definitions for the counter tile dimensions, which are based on the value of n'' and h , to form versions which represent 0 and 1 for each bit position. However, the geometries of the cap tiles are also dependent upon the binary value of s (padded to 8 bits), in this case "00011100", since they perform a mapping of counter values to bit values of s . In this case, the geometries of socket S_1 in the 0-cap, which has positions for all 8 possible counter values, must have openings in exactly the positions 0, 1, 5, 6, and 7 (and hence filled-in locations at positions 2, 3, and 4). Recall that the positions are numbered from north to south for the S_1 sockets. The 1-cap should have the exact opposite pattern of openings and filled-in locations for its socket S_1 , as should the socket $\overline{S_1}$ of the 0-cap. Clearly, the socket $\overline{S_1}$ of the 1-cap should match the 0-cap's S_1 socket.

The geometries of the S_2 sockets perform a mapping of counter values to bit values of s as well, but they perform a "one-off" mapping by matching bit values of s with the counter values which are 1 greater than the bit positions. Additionally, the direction in which the positions of the socket are labeled is reversed, meaning they are counted from the south to north. Therefore, the sockets S_2 for the 0-cap and $\overline{S_2}$ for the 1-cap should have openings at exactly positions 1, 2, 3, and 7 counting from top to bottom, while the sockets $\overline{S_2}$ for the 0-cap and S_2 for the 1-cap should have openings at exactly positions 4, 5, and 6. Note that the 0th position of S_2 sockets, due to the one-off mapping, is an invalid position which we always fill in (intuitively, because the counter value used to align with an S_1 socket is always 1 greater than the number represented by the column containing the socket). Please see Table 2 for a full listing of the socket patterns and Figures 17, 18, and 19 to view examples of counter and buffer columns.

E.5 Equivalence of disconnected 2-dimensional tiles and 3-dimensional tiles

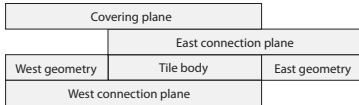


Figure 20: Side view of equivalent but connected 3-dimensional geometric tiles.

The construction in this section has been presented as a 2-dimensional construction meaning that the tiles are 2-dimensional objects, and due to the constraint of planarity they are never allowed to travel out of the plane during assembly. However, the geometries defined for the tiles often contain disconnected portions, namely filled-in units which are not connected via an unbroken path of additional filled-in units to their tile bodies. It is notable that by merely relaxing the restriction of forcing tiles to remain 2-dimensional, and instead allowing them to extend into three additional planes, this construction can achieve connectivity while removing the explicit restriction that translations of tiles during assembly must only occur within the $x - y$ plane since that restriction instead becomes implicit due to the design. See Figure 20

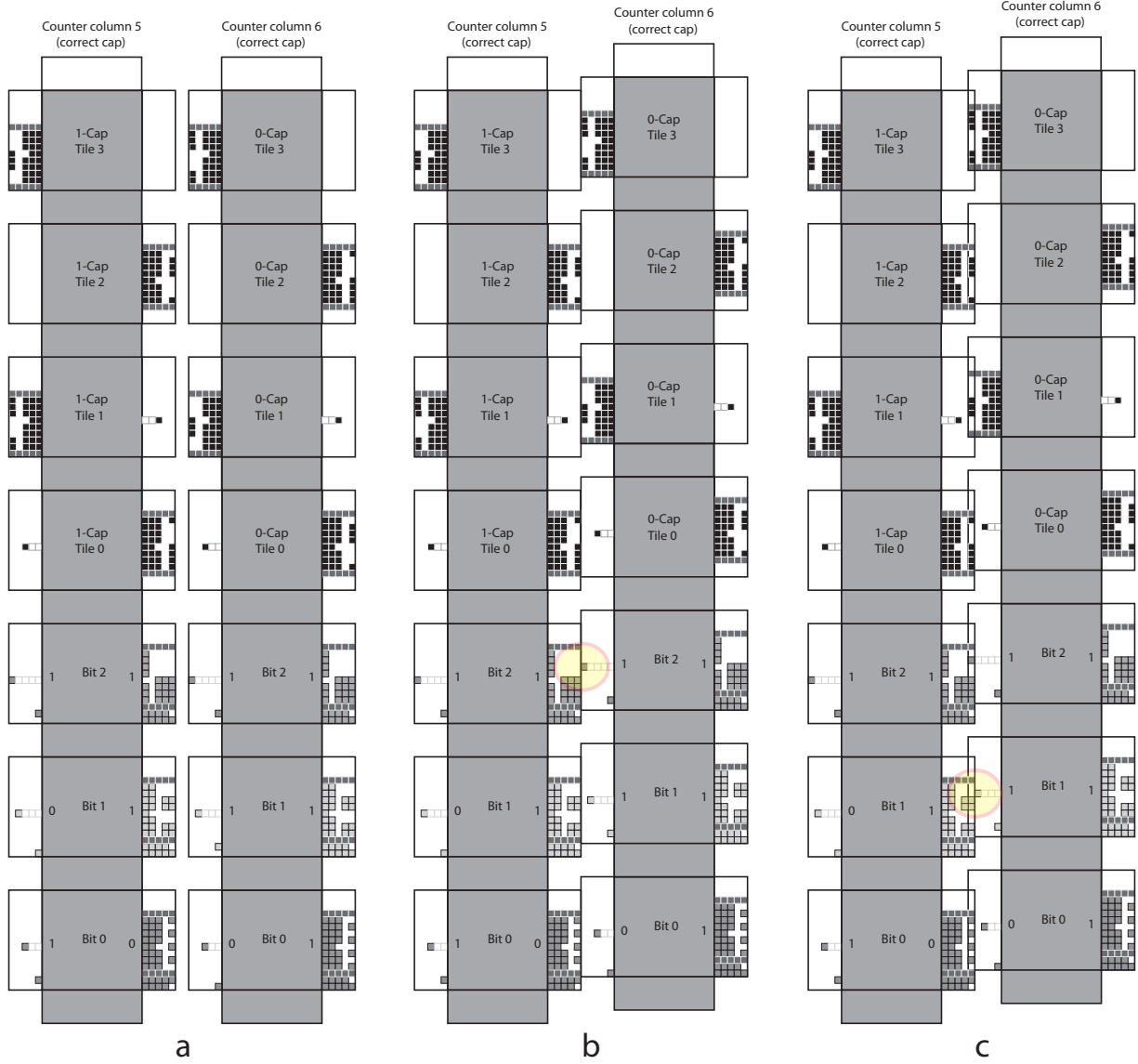


Figure 17: Two counter columns combining during the self-assembly of a 250×250 square (part 1). a) Two fully formed and correct counter columns representing the values 5 and 6, b) Translation of the right column to allow prong-to-socket alignment for the most significant bit (highlighted), and c) Translation for alignment of the second bit (highlighted).



Figure 18: Two counter columns combining during the self-assembly of a 250×250 square (part 2). a) After alignment of the final bit, all 4 prongs (highlighted) are able to slide into the back column of the sockets in the left column, b) Final north-south translation aligns each prong with the west-most gaps in each socket (highlighted) and also aligns the bit patterns of regions b_1 and b_2 of each pair of counter tiles (also highlighted), c) After the final east-west translation, the counter columns are fully combined, allowing the glues to interact and stably bind them.



Figure 19: Columns formed during the self-assembly of a 250×250 square. a) A counter column with the wrong cap, which prevents its combination with the next consecutive counter column, b. The incorrect counter column can combine with a buffer column encoding the correct counter value.

for a depiction of how the tiles are extended along the z -axis. For each of the east and west geometries a solid filled-in square of units is added in the plane above and below, respectively. These attach to every unit of the geometries as well as to additional filled-in squares above and below the tile body to ensure full connectivity. Finally, an additional layer is added which covers the tile body and lies above the west geometry, with exactly one open plane between it and the west geometry. In this way, it is ensured that all portions of the tiles and their geometries are connected, and exactly the tiles that could connect with each other in the 2-dimensional planar version can now connect with each other.

F Two Function Problem

In this section, we study the problem of bar-to-bump reduction for the design of the tile face geometries, namely converting bar geometries to bump geometries (see Section B). The goal is to attempt to simplify the geometry patterns as much as possible. Called the *two function problem*, it takes an input integer l , and requires the design of two functions $f : \{0, 1, \dots, l\} \rightarrow \{0, 1\}^n$ and $g : \{0, 1, \dots, l\} \rightarrow \{0, 1\}^n$ to satisfy $f(x) \text{ AND } g(y) = 0^n$ if and only if $x + y \leq l$, where *AND* takes as arguments two binary strings, x and y , of matching length and returns the string of binary values representing the logical AND operation between each corresponding pair of bits in x and y . The target is to make n as small as possible. It is trivial to design the two functions with $n = l$. We show a lower bound that exactly matches the upper bound.

Definition F.1. Let $l, n \in \mathbb{N}$. Define $f : \{0, 1, \dots, l\} \rightarrow \{0, 1\}^n$ and $g : \{0, 1, \dots, l\} \rightarrow \{0, 1\}^n$ where f and g take a number between 0 and l , inclusive, and return a binary string of length n such that $f(x) \text{ AND } g(y) = 0^n$ if and only if $x + y \leq l$. (Note that we define *AND* as the function which takes as arguments two binary strings, x and y , of matching length and returns the string of binary values representing the logical AND operation between each corresponding pair of bits in x and y . In this case, there must not be a 1 in the same position of both bit strings.) Goal: For a given l , find f and g such that n is minimal.

We have Theorem F.2 for a lower bound for the two function problem. This lower bound exactly matches the upper bound shown in Theorem F.3.

Theorem F.2. For each l , every solution for the two function problem needs $n \geq l$.

Proof. Assume that we have a solution for the two function problem. Let $f : \{0, 1, \dots, l\} \rightarrow \{0, 1\}^n$ and $g : \{0, 1, \dots, l\} \rightarrow \{0, 1\}^n$ such that $f(x) \text{ AND } g(y) = 0^n$ if and only if $x + y \leq l$.

For a string s in $\{0, 1\}^n$, define $C(s)$ to be the number of 1s in s . For example, $C(010011) = 3$. Define $D(f) = \sum_{i=0}^l C(f(i))$. For each string $s = a_1 a_2 \dots a_n$ in $\{0, 1\}^n$, define $G(s) = \{i : a_i = 1 \text{ and } 1 \leq i \leq n\}$. For each string $s = a_1 a_2 \dots a_n$ in $\{0, 1\}^n$, define $s[i] = a_i$.

Let the solution f and g satisfy that $D(f)$ is the largest for the least n . We claim that $G(f(0)) \subseteq G(f(1)) \subseteq G(f(2)) \subseteq \dots \subseteq G(f(l))$.

Assume that there is a $m \in \{0, 1, \dots, l\}$ with $G(f(m)) \not\subseteq G(f(m+1))$. Let i be the index such that $f(m)[i] = 1$ and $f(m+1)[i] = 0$. Let $f(m) = a_1 a_2 \dots a_n$ and $f(m+1) = b_1 b_2 \dots b_n$. Define function f' as follows: $f'(k) = f(k)$ for each $k \neq m+1$, and $f'(m+1) = b_1 \dots b_{i-1} a_i b_{i+1} \dots b_n = b_1 \dots b_{i-1} 1 b_{i+1} \dots b_n$. We claim that f' and g form a new solution for the two function problem.

For each y with $(m+1) + y \leq l$, we have $m + y < l$. Therefore, $f(m) \text{ AND } g(y) = 0^n$. This implies $a_i \cdot g(y)[i] = 0$. Furthermore, we also have $f(m+1) \text{ AND } g(y) = 0^n$. Therefore,

$$\begin{aligned} C(f'(m+1) \text{ AND } g(y)) &\leq C(f(m+1) \text{ AND } g(y)) + a_i \cdot g(y)[i] \\ &\leq 0 + 0 \\ &= 0. \end{aligned}$$

Thus, $C(f'(m+1) \text{ AND } g(y)) = 0$. Therefore,

$$f'(m+1) \text{ AND } g(y) = 0^n.$$

For each y with $(m+1) + y > l$, we have $f(m+1) \text{ AND } g(y) \neq 0^n$. Since $a_i = 1$, we have

$$\begin{aligned} C(f'(m+1) \text{ AND } g(y)) &\geq C(f(m+1) \text{ AND } g(y)) \\ &\geq 1. \end{aligned}$$

Therefore,

$$f'(m+1) \text{ AND } g(y) \neq 0^n.$$

Combining the last two cases, we have that f' and g form a new solution for the two function problem. We have $D(f) < D(f')$. This contradicts the fact that $D(f)$ is the largest for the same n .

Therefore, the solution f and g has that $G(f(0)) \subseteq G(f(1)) \subseteq G(f(2)) \subseteq \dots \subseteq G(f(l))$. It is easy to see $f(i) \neq f(j)$ for $0 \leq i < j \leq l$ because $f(i) \text{ AND } g(l-i) = 0^n$ and $f(j) \text{ AND } g(l-i) \neq 0^n$. Thus, we have that $G(f(i))$ is a proper subset of $G(f(i+1))$. Therefore, $n \geq l$. \square

Theorem F.3. The two function problem always has a solution with $n = l$.

Proof. Let $f(i) = 1^i 0^{l-i}$ and $g(l-i) = 0^i 1^{l-i}$. It is easy to verify that $f(x) \text{ AND } g(y) = 0^l$ if and only if $x + y \leq l$. \square

G Matrix Problem

In this section, we study the complexity of a matrix problem that is related to tile geometry design for compatibility specifications. An efficient solution for this problem can achieve reductions in geometry sizes.

Called the *matrix problem*, the goal is generating a binary matrix via vector inner products. Given a $0-1$ matrix M of size $m \times n$, find a list of vectors E_1, \dots, E_m and a list vectors W_1, \dots, W_n of length l for each vector such that $M(i, j) = 0$ if and only if $E_i \cdot W_j = 0$, where \cdot is the inner product of two vectors. The target is to minimize the length l , which is denoted by $L(M) = l$ for the least l . We show that for almost all $n \times n$ binary matrices M , $L(M) = \Theta(n)$. A submatrix M_i of M is characterized by (R_i, C_i) where R_i is a subset of row indices of M and C_i is a subset of column indices of M . In the case that all 1 entries are in submatrices $M_1 = (R_1, C_1), \dots, M_t = (R_t, C_t)$ that satisfy $R_i \cap R_j = \emptyset$ and $C_i \cap C_j = \emptyset$ for $i \neq j$, we show the equality $L(M) = L(M_1) + \dots + L(M_t)$. This relationship gives a tool for solving a class of concrete matrix problems. For another case that all 0 entries of M are in submatrices $M_1 = (R_1, C_1), \dots, M_t = (R_t, C_t)$ that satisfy $R_i \cap R_j = \emptyset$ and $C_i \cap C_j = \emptyset$ for $i \neq j$, we derive the lower bound and upper bound $\max(L(M_1), \dots, L(M_t)) \leq L(M) \leq (1 + \epsilon) \max(L(M_1), \dots, L(M_t)) + O(\frac{\log n}{\epsilon})$ for an arbitrary constant $\epsilon > 0$ using a randomized algorithm. This randomized algorithm can be used to find a matrix design with a small length l . The dimension of each submatrix M_i is $m_i \times n_i$ in table 1.

For two lists E_1, \dots, E_m and W_1, \dots, W_n such that each E_i or W_i are of length l , they are called a *l-list pair*.

Matrix Problem: Given a $m \times n$ $0-1$ matrix M , find a list vectors E_1, \dots, E_m and a list vectors W_1, \dots, W_n of length l for each vector such that $M(i, j) = 0$ if and only if $E_i \cdot W_j = 0$, where \cdot is the inner product of two vectors such that $E_i \cdot W_j = \sum_{k=1}^l e_k w_k$ for $E_i = e_1 e_2 \dots e_l$ and $W_j = w_1 w_2 \dots w_l$. Minimize the length l .

For an integer matrix $M_{m \times n}$, its rank is the number of linear independent rows. A matrix $M_{n \times n}$ is *diagonal one* matrix if all the diagonal elements are one and all other elements are zero. A matrix $M_{n \times n}$ is *diagonal zero* matrix is all diagonal elements are zero and all other elements are one.

G.1 Some Lower Bounds

In this section, we show some results for the lower bound of the matrix problem. Most of the lower bounds results are for concrete matrix problems.

Definition G.1. For a binary matrix M , define $L(M)$ to be the least length l for a solution of the matrix problem M .

Definition G.2. Assume that $M = (a_{i,j})_{m \times n}$ is a binary matrix.

- Define $S(M) = \min(m, n)$.
- A submatrix $M' = (R, C)$ of M is characterized by a set R of rows and a set C of columns in M . The elements of M' are all elements $a_{i,j}$ with $i \in R$ and $j \in C$. We also define $S(M') = \min(|R|, |C|)$.
- Two submatrices $M_1 = (R_1, C_1)$ and $M_2 = (R_2, C_2)$ of M are *independent* if $R_1 \cap R_2 = \emptyset$ and $C_1 \cap C_2 = \emptyset$.
- A set H of submatrices of M is *independent* if every two of them are independent.
- Let $H = \{M_1, \dots, M_t\}$ be a set of independent submatrices, define $T(H) = \max(S(M_1), \dots, S(M_t))$.
- For a sequence $s = a_1 a_2 \dots a_n$, and a subset $P = \{i_1, \dots, i_m\}$ with $i_1 < i_2 < \dots < i_m$ of integers of $\{1, 2, \dots, n\}$, define $s[P] = a_{i_1} a_{i_2} \dots a_{i_m}$. For an integer $i \leq n$, define $s[i] = a_i$.

Theorem G.3. Let M_1, \dots, M_t be independent submatrices of $M_{m \times n}$ and all 1s of M are in M_1, \dots, M_t . Then $L(M) = L(M_1) + \dots + L(M_t)$.

Proof. Assume that E_1, \dots, E_m and W_1, \dots, W_n form a solution of least length l for the matrix $M = (a_{i,j})_{m \times n}$. Let $M_i = (R_i, C_i)$ for $i = 1, \dots, t$. Define $P_i = \{j : \text{for some } (u, v) \in R_i \times C_i \text{ with } E_u[j] = W_v[j] = 1\}$.

Claim 1. $E_{r_1}[P_i], \dots, E_{r_y}[P_i]$ and $W_{c_1}[P_i], \dots, W_{c_z}[P_i]$ form a solution for M_i , where $R_i = \{r_1, \dots, r_y\}$ and $C_i = \{c_1, \dots, c_z\}$.

Proof. By the definition of P_i , for each $(u, v) \in R_i \times C_i$, we have $E_u[x] = 0$ or $W_v[x] = 0$ for any $v \notin P_i$ (otherwise, x is in P_i). Thus, $E_u \text{ AND } W_v = 0^l$ if and only if $E_u[P_i] \text{ AND } W_v[P_i] = 0^{l_i}$, where $l_i = |P_i|$. \square

Claim 2. $P_i \cap P_j = \emptyset$ for $i \neq j$.

Proof. This can be proved by contradiction. Assume $P_i \cap P_j \neq \emptyset$. Let $x \in P_i \cap P_j$. There is a $(u_1, v_1) \in R_i \times C_i$ such that $E_{u_1}[x] = W_{v_1}[x] = 1$. There is a $(u_2, v_2) \in R_j \times C_j$ such that $E_{u_2}[x] = W_{v_2}[x] = 1$. Therefore, $(E_{u_1} \text{ AND } W_{v_2})$ contains a bit $E_{u_1}[x] \cdot W_{v_2}[x] = 1$. Since M_1, \dots, M_t are independent, $u_1 \in R_i$ and $v_2 \in C_j$, we have $a_{u_1, v_2} = 0$. This contradicts that $(E_{u_1} \text{ AND } W_{v_2}) = 0^l$ if and only if $a_{u_1, v_2} = 0$. \square

Claim 3. $L(M) \geq L(M_1) + \dots + L(M_t)$.

Proof. By Claim 1, since $E_1[P_i], \dots, E_m[P_i]$ and $W_1[P_i], \dots, W_n[P_i]$ form a solution for the matrix problem M_i for $i = 1, \dots, t$, we have $L(M_i) \leq |P_i|$. By Claim 2, we have $|P_1| + \dots + |P_t| \leq l = L(M)$. Thus, $L(M) = |P_1| + \dots + |P_t| \geq L(M_1) + \dots + L(M_t)$. \square

Assume that $E_{i, r_1}, \dots, E_{i, r_y}$ and $W_{i, c_1}, \dots, W_{i, c_z}$ form a solution for $M_i = (R_i, C_i)$ with $R_i = \{r_1, \dots, r_y\}$ and $C_i = \{c_1, \dots, c_z\}$ for $i = 1, \dots, t$. The length of each vector for M_i is $l_i = L(M_i)$.

For $j = 1, \dots, m$, define $E'_j = A_1 \dots A_t$ such that $A_i = 0^{l_i}$ if $i \notin R_1 \cup \dots \cup R_t$, and $A_i = E_{u, r_h}$ if $i = r_h \in R_u$. For $j = 1, \dots, n$, define $W'_j = B_1 \dots B_t$ such that $B_i = 0^{l_i}$ if $i \notin C_1 \cup \dots \cup C_t$, and $B_i = W_{u, c_h}$ if $i = c_h \in C_u$.

Claim 4. E'_1, \dots, E'_m and W'_1, \dots, W'_n form a solution for M .

Proof. Let $l = l_1 + \dots + l_t$. Let's consider a position (i, j) in matrix M . Let $E'_j = A_1 \dots A_t$ and $W'_j = B_1 \dots B_t$. We discuss the following two cases.

- Case 1. $(i, j) \notin R_u \times C_u$ for all $u = 1, \dots, t$. For each $g \leq t$, we have either $A_g = 0^{l_g}$ or $B_g = 0^{l_g}$ since the submatrices M_1, \dots, M_t are independent. Therefore, $E'_i \text{ AND } W'_j = 0^l$.

- Case 2. $(i, j) \in R_u \times C_u$. Since the submatrices M_1, \dots, M_t are independent, we have $(i, j) \notin R_v \times C_v$ for $v \neq u$. We have $A_v \text{ AND } B_v = 0^{l_v}$ for each $v \neq u$. Thus, $E'_i \text{ AND } W'_j \neq 0^l$ if and only if $A_u \text{ AND } B_u \neq 0^{l_u}$.

□

By Claim 4, we have $L(M) \leq L(M_1) + \dots + L(M_t)$. By Claim 3, we have $L(M) = L(M_1) + \dots + L(M_t)$. □

Corollary G.4. There minimum length for the $n \times n$ diagonal-one matrix problem is exactly n .

Theorem G.5. For most of $n \times n$ matrices, the minimum solution is at least $\frac{n}{2} - n^{1-\epsilon}$ for any fixed $\epsilon > 0$.

Proof. Let ϵ be an arbitrary positive constant. There are totally 2^{n^2} many $n \times n$ binary matrices M . Assume $m \leq \frac{n}{2} - n^{1-\epsilon}$. We consider how many matrices can be constructed by using l -list pairs with $l \leq m$.

The total number of bits of a l -list pair is $2nl$. The total number of l -list pairs is 2^{2nl} . The total number of l -list pairs with $l \leq m$ is $\sum_{l=1}^m 2^{2nl} < 2 \cdot 2^{2nm} = o(2^{n^2})$ since $m \leq \frac{n}{2} - n^{1-\epsilon}$. Therefore, for most of $n \times n$ matrices, there is no $\frac{n}{2} - n^{1-\epsilon}$ solution.

□

Theorem G.6. For the minimum length solution for a full rank matrix is greater than $\log n$.

Proof. Assume that there is a l -list pair E_1, \dots, E_m and W_1, \dots, W_n solution for an $n \times n$ matrix M of rank n . If there are two different $i \neq j$ with $E_i = E_j$, the i -th row and j -row are the same. Thus, the matrix is not full rank. If $l = \log n$, then there is E_i to be all zeros. Thus the matrix M has a row to be all zeros. This makes M not to be full rank. Therefore, $l > \log n$.

□

Theorem G.7. For the minimum length solution for the $n \times n$ diagonal zero matrix is at greater than $\log n$.

Proof. The absolute value of the determinant of diagonal zero matrix is $(n-1)$. Thus, it is full rank matrix. It follows from Theorem G.6.

□

Theorem G.8. Assume that k is an integer such that there is another integer $1 < l_1 < l$ such that $\binom{l}{l_1} \geq n$. Then there is a l -list pair solution for the diagonal zero matrix problem.

Proof. Let E_1, \dots, E_n be n different binary string of length l with exactly l_1 ones each. Let W_i be the complementary binary string of E_i . It is easy to see that $E_i \cdot W_i = 0$ and $E_i \cdot W_j > 0$ for $i \neq j$.

□

Corollary G.9. The minimum length for diagonal zero $n \times n$ matrix is between $\log n + 1$ and $\log n + \log \log n$.

Proof. Assume that l is an even even number $\leq \log n + \log \log n$. Let $l_1 = l/2$. By Stirling formula $\frac{n!}{\sqrt{2\pi n}((\frac{n}{e})^n)} = 1$, we have $\binom{l}{l_1} \sim \frac{2^{l\sqrt{2}}}{\sqrt{\pi l}}$. Thus, we can pick a $l \leq \log n + \log \log n$ such that $\binom{l}{l_1} \geq n$. The $n \times n$ diagonal zero matrix is of rank n . The lower bound $\log n + 1$ follows from Theorem G.7.

□

G.2 Upper Bounds and Algorithm for Matrix Problem

In this section, we show a randomized algorithm to handle a class of matrix problems. A matrix $M_{m \times n}$ has a list of independent submatrices M_1, \dots, M_t that contain all zero entries of M . We derive an upper bound of the solution for $L(M)$ to be close to $\max(L(M_1), \dots, L(M_t))$. This implies the interesting bound $\max(L(M_1), \dots, L(M_t)) \leq L(M) \leq (1 + \epsilon) \max(L(M_1), \dots, L(M_t)) + O(\frac{\log(m+n)}{\epsilon})$ for an arbitrary positive constant ϵ .

We believe the following simple algorithm in Lemma G.10 for a random permutation is not new. For completeness, we include it here.

RandomPermutation(n)

Let $S = \{1, 2, \dots, n\}$;
 For i from 1 to n
 Select a random a_i element from S
 Let $S = S - \{a_i\}$;
 Output $a_1 a_2 \dots a_n$;
End of RandomPermutation

Lemma G.10. Every permutation of $1, 2, \dots, n$ has a equal probability to be generated by RandomPermutation(.) that runs in $O(n^2)$ time.

Proof. Assume that $a_1 a_2 \dots a_n$ be an arbitrary permutation. We just need to prove that $P_1 = a_1 a_2 \dots a_i a_{i+1} \dots a_n$ and $P_2 = a_1 a_2 \dots a_{i+1} a_i \dots a_n$ have the equal chance to be generated. This is because a permutation can be converted into another permutation via a finite number of swaps between two neighbor items. Note that P_1 can be converted into P_2 by swapping the two elements a_i and a_{i+1} . Assume that the partial permutation $a_1 a_2 \dots a_{i-1}$ have been generated by RandomPermutation(.). We have that $a_i a_{i+1}$ and $a_{i+1} a_i$ will be generated by a equal probability RandomPermutation(.) to append to the last partial permutation $a_1 a_2 \dots a_{i-1}$. The computational time follows from that fact that it takes $O(n)$ time to generate one element and update the set S in the algorithm. \square

Lemma G.11. Assume that S_1 is a subset of k_1 elements of $\{1, 2, \dots, n\}$. Let S_2 be a subset of $\{1, 2, \dots, n\}$ and of size k_2 with $k_2 \geq \alpha n$ for some $\alpha \in (0, 1)$. Then with probability at most $p \leq (1 - \alpha)^{k_1}$, $P[S_2] \cap S_1 = \emptyset$ for a random permutation P of $1, 2, \dots, n$, where $P[S_2]$ is the set of the positions of elements S_2 in P .

Proof. For two equal size subsets S and S' of $\{1, 2, \dots, n\}$, the number of permutations P that make $P[S_2] \cap S = \emptyset$ is the same as the number of permutations P that make $P[S_2] \cap S' = \emptyset$. By Lemma G.10, the probability for $P[S_2] \cap S = \emptyset$ is the same as the probability for $P[S_2] \cap S' = \emptyset$, where P is a random permutation of $1, 2, \dots, n$. Thus, we can assume that S_1 is $\{1, 2, \dots, k_1\}$.

The probability that none of the elements of S_2 are selected in the first k_1 positions is

$$p = \frac{n - k_2}{n} \cdot \frac{n - k_2 - 1}{n - 1} \dots \frac{n - k_1 - k_2 - 1}{n - k_1 - 1}.$$

Consider the function $f(x) = \frac{n - k_2 - x}{n - x} = 1 - \frac{k_2}{n - x}$. We have its derivative function $\frac{\partial f(x)}{\partial x} = -\frac{k_2}{(n - x)^2} < 0$. Thus, $f(x)$ is a decreasing function. We have $\frac{n - k_2 - i}{n - i} \leq \frac{n - k_2}{n}$. Thus,

$$\frac{n - k_2}{n} \cdot \frac{n - k_2 - 1}{n - 1} \dots \frac{n - k_1 - k_2 - 1}{n - k_1 - 1} \leq \left(\frac{n - k_2}{n}\right)^{k_1} \quad (1)$$

$$\leq (1 - \alpha)^{k_1}. \quad (2)$$

\square

Lemma G.12. There is algorithm that given a binary matrix $M_{m \times n}$, it gives a solution for the generalized matrix problem with $S(M)$ length in $O(mn)$ times, where $S(M) = \min(m, n)$ as in Definition G.2.

Proof. Let $M = (a_{i,j})_{m \times n}$. Without loss of generality, assume $n \leq m$. For each column k , let $W_k = 0^{k-1} 1 0^{n-k}$. For each row j , let $E_j = e_{j,1} \dots e_{j,n}$, where $e_{j,k} = 1$ if and only if $a_{j,k} = 1$. \square

Definition G.13. For a string s , define a padding function $pad_1(s, k_1, k_2, k_3) = 0^{k_1} 1^{k_2} s 1^{k_3 - |s|}$ if $|s| \leq k_3$, and $pad_1(s, k_1, k_2, k_3)$ is the empty string otherwise. We also define another padding function $pad_2(s, k_1, k_2, k_3) = 1^{k_1} 0^{k_2} s 1^{k_3 - |s|}$ if $|s| \leq k_3$, and $pad_2(s, k_1, k_2, k_3)$ is the empty string otherwise. For a permutation $p = i_1 \dots i_n$ of $1, \dots, n$, and a binary string $s = a_1 \dots a_n$ of length n , define $P(s, p) = a_{i_1} \dots a_{i_n}$.

Lemma G.14. Let ϵ be an arbitrary constant in $(0, 1)$. Then there is an $O((m+n)^3)$ time randomized algorithm such that given a binary matrix $M_{m \times n}$, a set $H = \{M_1, \dots, M_t\}$ of independent submatrices that all zeros entries of M are in the submatrices of H , and also a solution with length l_i for each $M_i \in H$, it produces a solution for M with length $(1 + \epsilon)U(H) + O(\frac{\log(n+m)}{\epsilon})$, where $U(H) = \max\{l_i : M_i \in H\}$ is the largest length of the solution among all submatrices in H .

Proof. Let $k_1 = k_2 = \frac{\epsilon}{2}U(H) + \frac{\epsilon}{\epsilon} \log(m+n)$ and $k_3 = U(H)$, where constant c is selected such that

$$(1 - \frac{\epsilon}{4})^{\frac{\epsilon}{\epsilon} \log(n+m)} \leq \frac{1}{4mn}. \quad (3)$$

The total length for designing the matrix solution is $z = k_1 + k_2 + k_3$. We design E_1, \dots, E_m and W_1, \dots, W_n of length z each for a solution for the matrix problem M . By Lemma G.12, we assume $U(H) \leq \min(m, n)$.

For each submatrix in $M_u = (R_u, C_u)$ in H with $R_u = \{i_1, \dots, i_s\}$ and $C_u = \{j_1, \dots, j_t\}$, let M_u have a solution $E_{i_1}^{(u)}, \dots, E_{i_s}^{(u)}$ and $W_{j_1}^{(u)}, \dots, W_{j_t}^{(u)}$, which has length $l_u = |E_{i_1}^{(u)}| \leq U(H)$. Let p_u be a random permutation $1, 2, \dots, z$. Now let $E_{i_k} = P(\text{pad}_1(E_{i_k}^{(u)}), p_u)$ for $k = 1, \dots, s$ and let $W_{j_k} = P(\text{pad}_2(W_{j_k}^{(u)}), p_u)$ for $k = 1, \dots, t$. Thus, if row i and column j have $(i, j) \in R_u \times C_u$ for some $M_u = (R_u, C_u) \in H$, we have $E_i^{(u)} \text{ AND } W_j^{(u)} = 0^z$ if and only if $P(\text{pad}_1(E_{i_k}^{(u)}), p_u) \text{ AND } P(\text{pad}_2(W_{j_k}^{(u)}), p_u) = 0^z$ if and only if $E_i \text{ AND } W_j = 0^z$ if and only if $a_{i,j} = 0$.

For each row i , if $i \notin R_k$ for any $M_k = (R_k, C_k) \in H$, then let $E_i = 1^z$. Similarly, for each column j , if $j \notin C_k$ for any $M_k = (R_k, C_k) \in H$, then let $W_j = 1^z$. Thus, for row i and column j , if $i \notin R_k$ for any $M_k = (R_k, C_k) \in H$, we always have $E_i \text{ AND } W_j \neq 0^z$. Similarly, if $j \notin C_k$ for any $M_k = (R_k, C_k) \in H$, we always have $E_i \text{ AND } W_j \neq 0^z$.

Let

$$\alpha = \frac{k_1}{z} \quad (4)$$

$$= \frac{k_1}{2k_1 + U(H)} \quad (5)$$

$$= \frac{\frac{\epsilon}{2}U(H) + \frac{\epsilon}{\epsilon} \log(m+n)}{2(\frac{\epsilon}{2}U(H) + \frac{\epsilon}{\epsilon} \log(m+n)) + U(H)} \quad (6)$$

$$\geq \frac{\epsilon U(H) + \frac{2c}{\epsilon} \log(m+n)}{2\epsilon U(H) + 2U(H) + \frac{4c}{\epsilon} \log(m+n)} \quad (7)$$

$$\geq \frac{\epsilon U(H) + \frac{2c}{\epsilon} \log(m+n)}{4U(H) + \frac{4c}{\epsilon} \log(m+n)} \quad (8)$$

$$\geq \frac{\epsilon U(H) + \epsilon \cdot \frac{c}{\epsilon} \log(m+n)}{4U(H) + \frac{4c}{\epsilon} \log(m+n)} \quad (9)$$

$$= \frac{\epsilon}{4}. \quad (10)$$

For an entry (i, j) , we consider the case that $i \in R_u$ in $M_u = (R_u, C_u)$, and $j \in C_v$ in $M_v = (R_v, C_v)$ for some $u \neq v$. For a sequence $s = a_1 \dots a_z$, let $Q_1(s)$ be the the set of positions with bit 1 in s ($Q_1(s) = \{i : a_i = 1 \text{ and } 1 \leq i \leq z\}$). Let $E_{i_1}^{(u)}, \dots, E_{i_s}^{(u)}$ and $W_{j_1}^{(u)}, \dots, W_{j_t}^{(u)}$ be a solution for submatrix M_u with length $l_u = |E_{i_1}^{(u)}| \leq U(H)$. Since $i \in R_u$, $E_i = P(\text{pad}_1(E_{i_k}^{(u)}), p_u)$, where p_u is a random permutation of $1, 2, \dots, z$, and $i_k = i$. Let $E_{x_1}^{(v)}, \dots, E_{x_a}^{(v)}$ and $W_{y_1}^{(v)}, \dots, W_{y_b}^{(v)}$ be the solution for submatrix M_v with length $l_v = |E_{i_1}^{(v)}| \leq U(H)$. Since $j \in C_v$, $W_j = P(\text{pad}_2(W_{y_k}^{(v)}), p_v)$, where p_v is a random permutation of $1, 2, \dots, z$, and $y_k = j$. Since $u \neq v$, p_u and p_v are independent permutations. $Q_1(E_i)$ contains k_1 positions of $\{1, 2, \dots, z\}$, and $Q_1(W_j)$ another k_2 positions of $\{1, 2, \dots, z\}$. By Lemma G.11, with probability at most $(1 - \alpha)^{k_2}$, $Q_1(E_i) \cap Q_1(W_j) = \emptyset$. Therefore, with probability at most $(1 - \alpha)^{k_2}$, $E_i \text{ AND } W_j = 0^n$.

Therefore, with probability at most $p = nm(1-\alpha)^{k_2}$, there is a position (i, j) in M with E_i AND $W_j = 0^z$ not to be equivalent to $a_{i,j} = 0$. By the choice of c at equation (3), and inequalities (4) to (10), we have small probability p defined above to be at most $\frac{1}{4}$.

The computational time follows from the fact that for each E_i or W_j , we need at most $O((m+n)^2)$ time, which is spent for generating a random permutation, from the solutions of matrices in H . \square

Theorem G.15. Let ϵ be an arbitrary constant in $(0, 1)$. Assume that H is a set of independent submatrices of M that all zero elements in M are in the submatrices of H , then $V(H) \leq L(M) \leq (1 + \epsilon)V(H) + O(\frac{\log(n+m)}{\epsilon})$, where $V(H) = \max\{L(M_i) : M_i \in H\}$.

Proof. It is trivial to see the inequality $V(H) \leq L(M)$ since a solution for M automatically implies a solution for its submatrix. The inequality $L(M) \leq (1 + \epsilon)V(H) + O(\frac{\log(n+m)}{\epsilon})$ follows from Lemma G.14. \square

Theorem G.16. Let ϵ be an arbitrary constant in $(0, 1)$. There is a randomized algorithm such that given a binary matrices $M_{m \times n}$ and a set H of independent submatrices of M , if all zero elements M are in the submatrices of H , then the algorithm returns a design with total length at most $(1 + \epsilon)T(H) + O(\frac{\log(n+m)}{\epsilon})$. Furthermore, the time complexity of the algorithm is $O((n+m)^3)$.

Proof. It follows from Lemma G.12 and Lemma G.14. \square

Lemma G.17. There is a $O(2^{L(M) \min(m,n)}(m+n)^3)$ time algorithm to find an optimal solution of length $L(M)$ for a binary matrix $M_{m \times n}$.

Proof. By Lemma G.12, it has a solution of length at most $\min(m, n)$. Assume $m \leq n$. We can find the least length by trying all numbers at most $L(M)$. Each number takes at most $O(2^{L(M)m})$ time. After vectors E_1, \dots, E_m are fixed, it takes $O(mn)$ time to derive each W_i . This can be done to put the least number of zeros in the W_i to satisfy the zero entries of column i of $M_{m \times n}$. Thus, it takes $O(mn^2)$ time to derive all the columns after fixing rows E_1, \dots, E_m . Therefore, the total time is $O(2^{L(M)}(m+n)^3)$. \square

Theorem G.18. Let ϵ be an arbitrary constant in $(0, 1)$. Then there is a randomized algorithm such that given a binary matrices $M_{m \times n}$ and a set H of independent submatrices of M , if all zero elements M are in the submatrices of H , then the algorithm returns a design with total length at most $(1 + \epsilon)V(H) + O(\frac{\log(n+m)}{\epsilon})$, where $V(H) = \max\{L(M_i) : M_i \in H\}$. Furthermore, the time complexity of the algorithm is $O(2^{V(H) \min(m,n)}(n+m)^3)$.

Proof. It follows from Lemma G.17 and Lemma G.14. \square

G.3 Revised Matrix Problem

In this section, we revise the definition of the matrix problem, and show that how the optimal length of solution depends on the rank of the matrix.

Revised Matrix Problem: Given a $n \times n$ nonnegative integer matrix M , find a list vectors E_1, \dots, E_n and a list vectors W_1, \dots, W_n of length l for each vector such that $M(i, j) = E_i \cdot W_j$, where \cdot is the inner product of two vectors. Minimize the length l .

Theorem G.19. Let M be a $n \times n$ matrix with nonnegative elements. The minimum length solution for the revised matrix problem M is at least $\text{rank}(M)$.

Proof. Assume that there is a l -list pair E_1, \dots, E_n and W_1, \dots, W_n solution M such that $M(i, j) = E_i \cdot W_j$. Define M_t to be the matrix such that $M_t(i, j) = E_i[t]W_j[j]$. M_t is generated by the t -th bit of the l -list pair. We have $M = M_1 + M_2 + \dots + M_l$. It is easy to see that rank of each M_t is at most one.

By the well known fact in the linear algebra, we have $\text{rank}(M) \leq \text{rank}(M_1) + \text{rank}(M_2) + \dots + \text{rank}(M_l) \leq l$. Therefore, $\text{rank}(M) \leq l$. \square

Corollary G.20. The minimum length solution for the full rank revised matrix with matrix size $n \times n$ problem is n .

Corollary G.21. The minimum length solution for the revised matrix problem with matrix size $n \times n$ for diagonal one matrix is n .

Corollary G.22. The minimum length solution for the revised matrix problem for diagonal zero $n \times n$ matrix is n .

Corollary G.22 is in contrast to Corollary G.9.

Proof. The determinant of a diagonal zero matrix is $n - 1$. It is full rank matrix. It follows from Theorem G.19.

□